AFRL-AFOSR-UK-TR-2014-0008

ROBIL - Robot Israel

Hugo Guterman

Ben Gurion University of the Negev (BGU)
Department of Electrical and Computer Engineering
1 Ben-Gurion Boulevard
Beer-Sheva 84105
ISRAEL

EOARD Grant 12-2143

Report Date: July 2013

Final Report from 30 September 2012 to 29 July 2013

Air Force Research Laboratory
Air Force Office of Scientific Research
European Office of Aerospace Research and Development
Unit 4515, APO AE 09421-4515

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
31 July 2013	Final Report	30 September 2012 – 29 July 2013

4. TITLE AND SUBTITLE

ROBIL - Robot Israel

5a. CONTRACT NUMBER

FA8655-12-1-2143

5b. GRANT NUMBER

Grant 12-2143

5c. PROGRAM ELEMENT NUMBER

61102F

6. AUTHOR(S)

Hugo Guterman

5d. PROJECT NUMBER

5d. TASK NUMBER

5e. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Ben Gurion University of the Negev (BGU)
Department of Electrical and Computer Engineering
1 Ben-Gurion Boulevard
Beer-Sheva 84105
ISRAEL

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

EOARD
Unit 4515
APO AE 09421-4515

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/AFOSR/IOE (EOARD)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

AFRL-AFOSR-UK-TR-2014-0008

12. DISTRIBUTION/AVAILABILITY STATEMENT

Distribution A: Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The objective of this research project was to develop and provide a dedicated man-machine interaction method tailored for supervised autonomous systems. Implementation of an optimal methodology for human-robot cooperation involved combining the advantages of human perception and recognition skills with the autonomous robots' accuracy and consistency. A cooperative human–robotic system can increase identification capabilities, and enable recognition even in unpredictable conditions that autonomous systems are incompetent to deal with. Providing means for optional, high-level human intervention, along with pre-acquired machine skills, stands at the base of our supervised autonomy control concept. The expected impact is to enable smooth real-time adaptation of the combined human–robot system to many possible changes of the environment and performances of both robot and human operator.

15. SUBJECT TERMS

EOARD, Robotics

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18, NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			James Lawton
UNCLAS	UNCLAS	UNCLAS	SAR	49	19b. TELEPHONE NUMBER *(Include area code)* +44 (0)1895 616187

Cover Sheet

• DARPA BAA-12-39

• DARPA Robotics Challenge - Track B

• Lead organization: Ben-Gurion University of The Negev, Israel (BGU)

• Type of business: Foreign Concern/Entity

• All other team members:

Participant organization name	Type Of Business	Part. short name
Technion – Israel Institute of Technology	Research University	IIT
Israel Aerospace Industry	Industry	IAI
Bar-Ilan	Research University	BIU
Cogniteam	Small Company	Cogniteam

• Research Title: ROBIL (Robot Israel)

• Principal Investigator:

Prof. Guterman Hugo
Department of Electrical and Computer Engineering, Ben-GurionUniversity
POB 653, Beer-Sheva, Israel, 84105
Email: hugo@ee.bgu.ac.il & hugo.guterman@gmail.com

• Administrative point of contact:

Pinto Dana
Research & Development Authority, Ben-GurionUniversity
1 Ben-Gurion Blvd., P.O Box 653, Beer-Sheva, 84105, ISRAEL
Tel.: 972-8-6477748; Fax: 972-8-6472930
dpinto@bgu.ac.il

• AWARD NO. FA8655-12-1-2143

• Place(s) and period(s) of performance:
Universities and Companies in Israel 30 September 2012– 31 July 2013

Table of Contents

List of Figures

1 Introduction

This report summarizes Robil's activities during the period of January - April 2013.
Robil's team efforts focused in the following areas:

- *Infrastructure Organization:* including definition of the software interfaces, software testing and debugging.
- *Operational:*Functional Analysis of the three events
- *Perception:* Five modules have been defined and partially implemented
- *Decision Making:*An execution engine that allows partial plans (behavior trees) to be composed into full plans that can be executed at runtime has been implemented
- *Dismounted Mobility:* four walking modes have been developed to facilitate crossing different types of terrain. Two modes, the continuous dynamic walk and the pentad walk allowed successful completion of task-2 as demonstrated in the video available at**:** **http://www.youtube.com/watch?v=TQwOfpzHg7c** and further detailed in section 3.4.1. A dynamic method for entering the car has been developed as demonstrated in the video available at: **http://www.youtube.com/watch?v=8QD9-YUzhng**and further detailed in section 3.4.2. The modular software architecture that supports the different walking modes is detailed in section 3.4.3.
- *Mounted Mobility:* Driving capabilities have been implemented and tested. Video is available at: http://www.youtube.com/watch?v=82Oj_V4PF80
- *Dexterity:* focus on single arm motion and grasping strategies and telerobotic operation.

A follow-up working workshop took place at the Technion in March 20, 2013.
Robil team has been working in fulfilling the task requirements for the qualifiers.

In parallel the integration has continued and is tested into the CloudSim environment.

1 Methods, Assumptions, and Procedures

1.1 Assumptions

Our assumptions are based upon the details published by DARPA's official releases of the VRC Rules and VRC Technical Guide document. We had also used the DRC forum and FAQ to dig for information as many of the details were unknown or not final during the whole period of design and development of the software.

One major assumption which is necessary to emphasize is that no team may compete with simulator parameters different than those of the official Gazebo. That is, no one can optimize solver step-size or number of iterations to fit his needs.

1.2 Methods

1.2.1 System Capabilities

Utilizing ROS architecture, we have denoted a set of well-defined capabilities which in different compositions forms complex behaviors. Each capability shall be implemented as a ROS package by its designated group, and react as autonomous unit of the complete system, and with the ability to request services from other packages. A definition of all the capabilities relevant for the VRC can be found in Table 1.

TABLE 1: ROBIL SYSTEM CAPABILITIES DEFINED FOR VRC

Task	Symbol	Capability Name	Description
T1 Operation	C11	Operator control	HMI unit for event control by intervention in mission planning and execution (T3) processes and level of autonomy.
		Agent	Used as a gateway between the OCU and the robot at the robot's side. Deliver all messages to and from the OCU and controls the communications.
T2 Perception	C21	Vision and Lidar	Operate camera and Lidar to capture scene at a requested azimuth.
	C22	Ground Recognition and Mapping	get ground position and surface shape
	C23	Object recognition	Search for a specific object in an image
	C24	Obstacle Detection	Detect obstacles on ground and above ground
	C25	Global Position	Find robot relative position in global map and process IMU data
T3 Decision-Making	C31	Path Planning	Search for a global route on a map
	C34	Mission planner and executor	Behavior Trees planning and execution
	C35	Mission Monitoring	Monitoring task progress by condition validation
T4 Dismounted Mobility	C41	Quasi Static control	1. Foot placement; 2. Single step; 3. Turn in place
	C42	Dynamic Locomotion	Walk by requested path\direction and maintain stability all times. 4 modes: 1. Continuous Dynamic; 2. Discrete Foot Placement

			(including FP planning); 3. Panted walk; 4. Translation \ Rotation behavior
	C45	Posture Control	Control head and torso movements
	C46	Mount Vehicle	Get the robot from standing outside the vehicle to a safe sitting position inside it
	C47	Dismount Vehicle	Get the robot from sitting in the vehicle to a standing position outside it.
	C48	Stand Up	Stand up robot in case of falling
T5 Mounted Mobility	C51	Car operation	Drive and navigate the car based on vision.
T6 Dexterity	C65	Connect hose to spigot (high level dexterity)	Plan and follow procedure to connect hose to spigot
	C66	Grasp and release an Object (low level dexterity)	Operate hand to hold an object and release by command.
	C67	Car Manipulation (high level dexterity)	Perform manipulations by request from C51 to drive car (car ignition, steering wheel and gear)

1.3 Integration

The integration was responsible of making sure that interfaces are properly defined and then that the implementation matches needed functionality.

For ROBIL project, several teams were working distributed and in parallel. Each team was responsible of one or more ROS packages. The integration checks that ROS packages are interacting as defined in system functional requirements.

We used the following tools: git, github, jenkins

The workflow for integrating a package is as follows:

1. The official code repository is in the github (known as official tree).
2. Each team has a corresponding repository in the github (known as team repository)
3. The code to integrate should be in the local git repository
4. The code has to be pushed to the team repository in the github and a pull request has to be opened to integrator.
5. Jenkins server is then activated:
6. Pull the code
7. Compile it.
8. Perform sanity tests
9. Integrator then pull the new code and makes its own checks: functionality, bugs correction, integration
10. Issues are opened and tracked via trac server
11. If the new code is good enough, it is pushed to the official tree.

Debugging sessions are conducted with Google hangout since teams are geographically spread over the country.
We have added the cloudsim component of the tests:

1. 1. Jenkins server executes tests there on the field computer while the simulator computer is running the drcsim.
2. A script analyzes the logs in order to determine if tests have passedFigure 1 presents the integration and development Gantt of the last quarter in regard tothe program schedule.

2 Results and Discussion

2.1 Operation

2.1.1 Operational Concept Definition (OCD) summary

- ROBIL looks for a mission-specific set of data (for example car, door, steering wheel to find driver seat of a car) and when the set is found, it is sent to the operator as vector data over visual data for approval/correction.
- Minimal User intervention is achieved by ROBIL transmitting back visual information with perceived mission plan in predefined decision points or when no solution is found.
- Only mission data (path, object recognition etc.) or predefined action command sequences are transmitted to ROBIL by the operator.
- Tethering shall only be used when no solution is found and only per specific segment (for example – tethered hand operation while keeping stability and mobility modules autonomous).
- Behavior tweaking is done using sliders of shifting the weight of different variables to encourage Faster/Safer operation, more/less feedback and more/less autonomy.
- A bank of preset motion sequences and sensor operations shall be available to the operator at all times (stop, kneel, sit, get image, get obstacle map etc.)
- Ground rules of operation:
    - No operator override for 10 seconds = approval
    - Operator intervention starts = ROBIL stops and waits
    - Operator transmission shall be answered - Received and Done

2.1.2 Implemented Functionality:

- Task selection
- Execution control (run, pause, resume)
- Receiving images
- Receiving occupancy grid
- Receiving path
- Operator requests
- Override and draw new path
- Robot status, location and orientation
- View updated Behavior tree

2.1.3 HMI architecture

The architecture of the C11 module consists of two units - C11 Operator Control, the OCU computer at the operator side, and C11 Agent which is at the robot side (Figure 2).

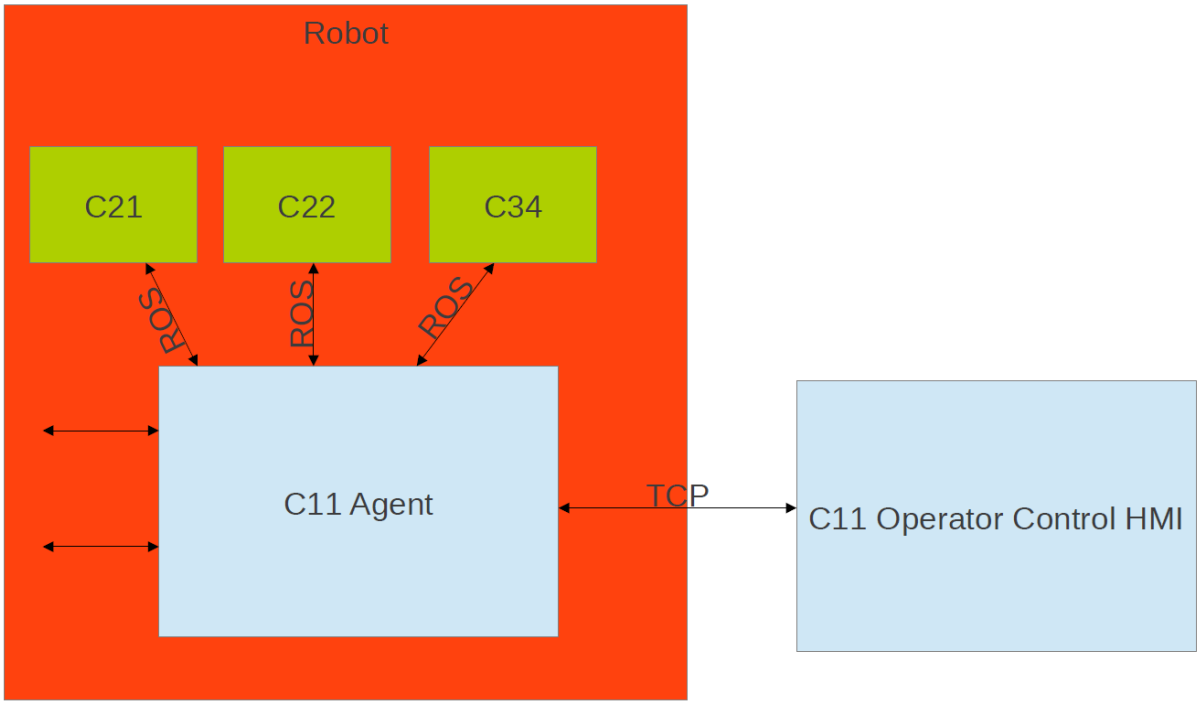


FIGURE 3.1.1: HMI ARCHITECTURE

2.1.4 OCU Design

Figure 3 demonstrates the GUI at the OCU unit. The GUI is divided into 3 segments:

- Left: Mission control- where the operator can tweak the robot's behavior, control autonomy levels and request for information.
- Middle: Image bank
- Right: Occupancy grid with overlaying vector data of robot position, orientation and the planned path. The operator may change robot's scene understanding by adding or editing vector data.

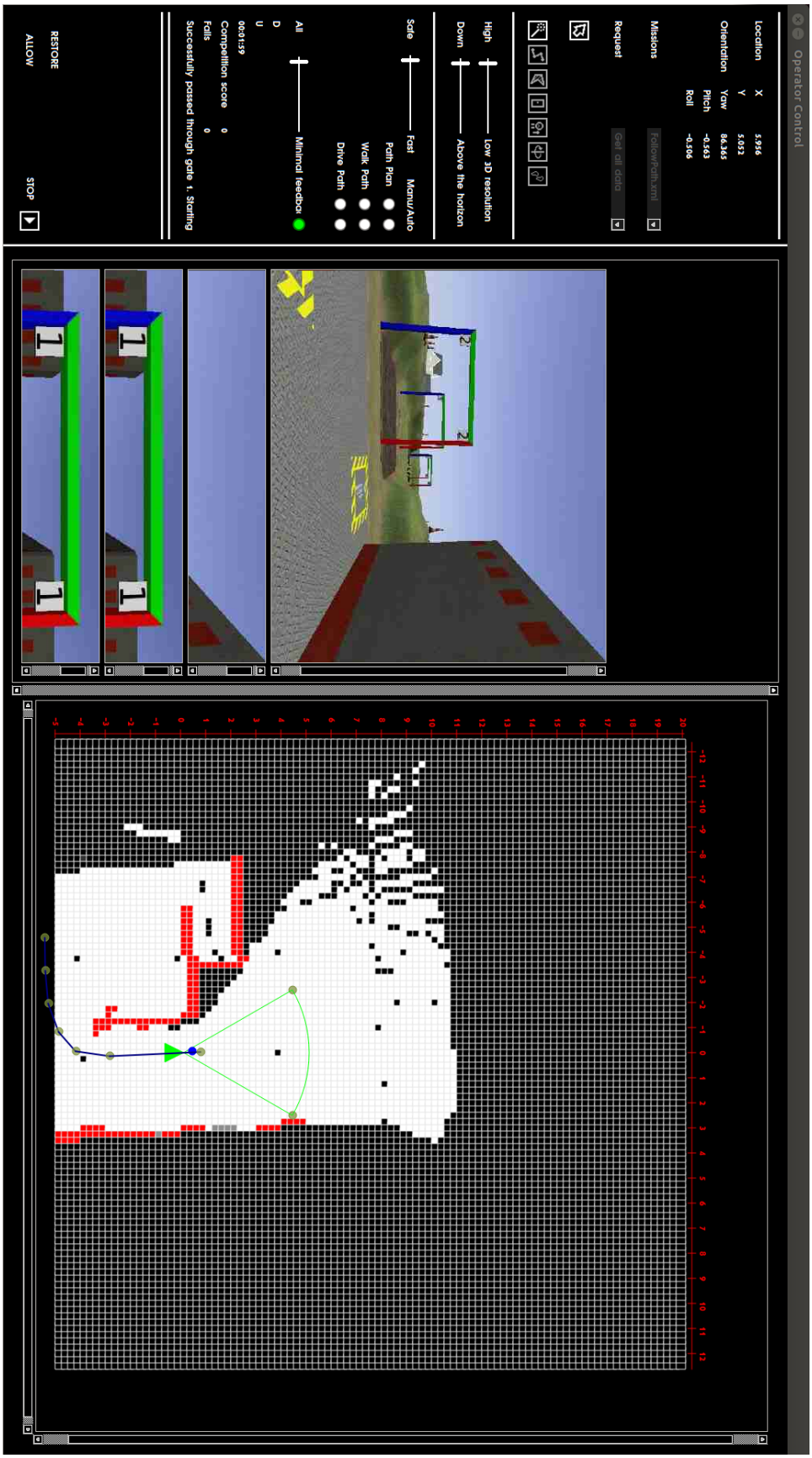


FIGURE 3.1.2: OCU: GUI design

2.1.5 Operation at the VRC

The final operation concept, which was used at the VRC runs, was based upon the understanding that much of our planned autonomous capabilities weren't finalized. After loading the mission, the operator usually requested a picture and obstacle grid from the robot. On the received grid the operator draw a path to get out from the starting pen, sent it to the robot and waited for a confirmation when done. The next step was to request a picture and grid again, and upon the current task to continue with giving paths and running modules to enter vehicle, crawl into mud or pick up the hose. In case that ATLAS fell down, a stand-up script was launched.

2.2 Perception

The workload was distributed between five nodes (Figure 3.2.1)

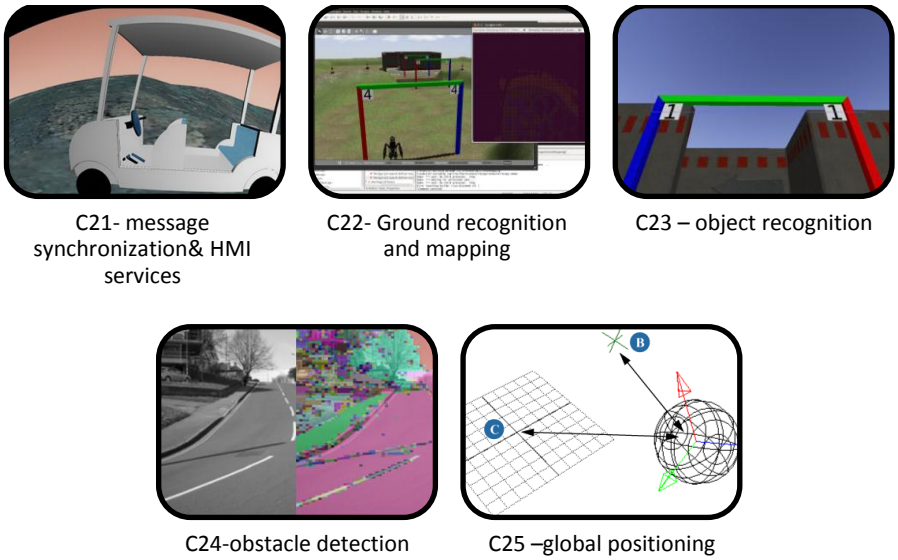


FIGURE 3.2.1: The five perception nodes

2.2.1 Module C21 – Vision and Lidar

Introduction

This module is used to synchronize the Cameras & Point cloud &Lidar input and act as a data server for data analyzing to be used by the other perception modules.

Method&Implementation

Using the message filters feature it synchronizes the cameras and the cloud and Lidar data together. The TF server is used to refer the data relative to the pelvis origin.

2.2.2 Module C22 – Ground Recognition and Mapping

Introduction

This module receives 3D Point Cloud and returns a matrix which represents an eagle-eye view of the terrain. This map is divided into $25cm^2$squares, each holding data on the

properties of the terrain, e.g. the square in the location $a_{i,j}$represents a floor, etc. The map information is used by the path planning module for mounted and dismounted mobility tasks.

Method

The module receives a 3D Point cloud from module C21 which is filtered using a VoxelGrid Filter. This process will leave only a relative amount of points in order to make the ground mapping process more efficient.

The filtered points are now divided into different groups of points in a stage called Segmentation. Planes are then calculated from these groups using RANSAC methodology. These planes consist of attributes such as slope, height and availability (floor or obstacle) and are mapped on the matrix according to their geometric position.

For fast mapping the Lidar data is filtered and transformed to a fixed frame. This way we achieve fast height mapping to go along with the plane detection

Implementation

Open source code from the PCL library was utilized in order to perform data operations on the given point cloud, such as filtering and plane segmentation using RANSAC.

For the filtering stage, VoxelGrid filtering script is employed and applied a square threshold of $5cm^2$, after filtering using this method a significantly smaller amount of points is received. Although fewer points are employed, it is possible to achieve an impressively accurate result.

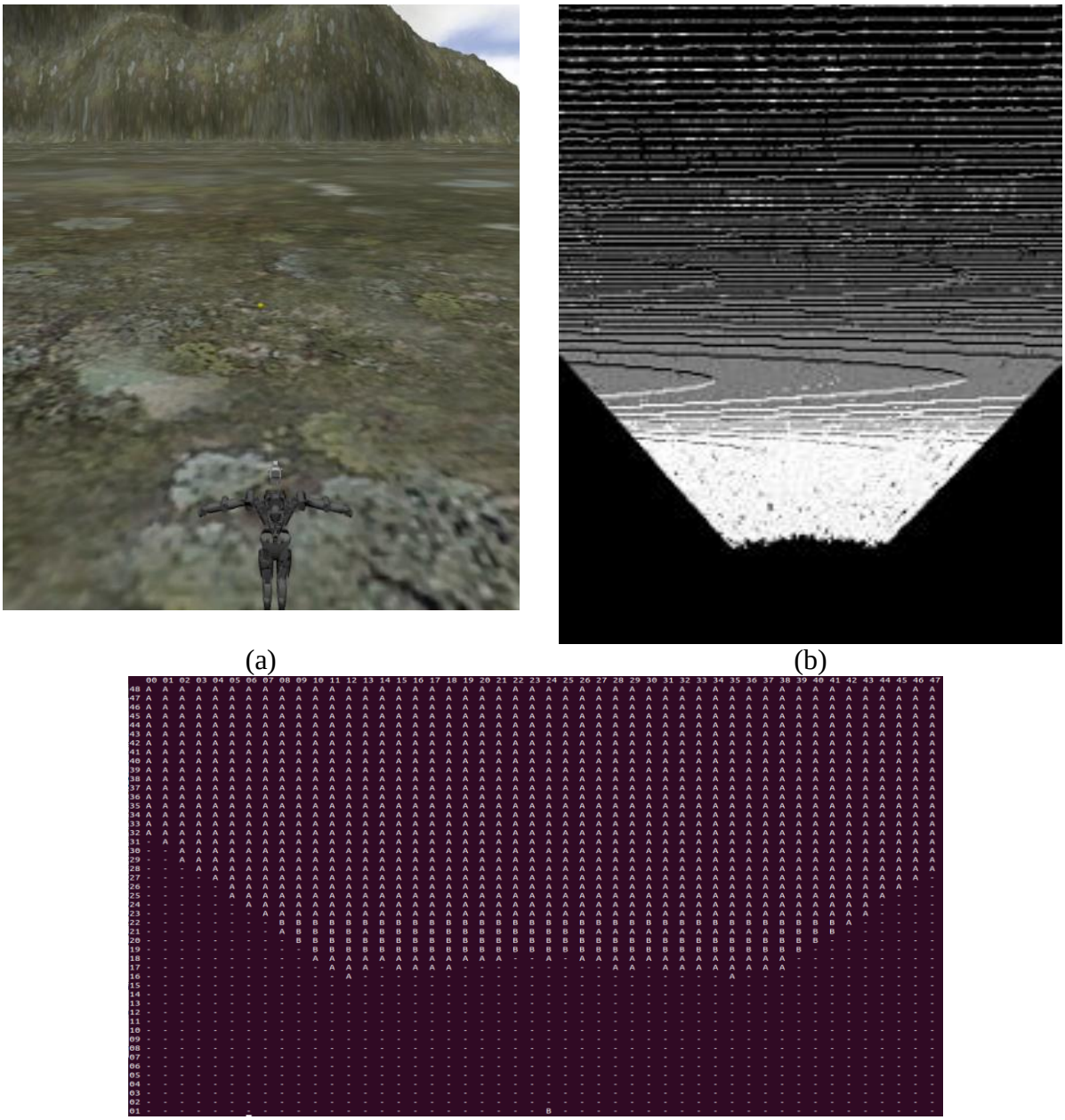
For the segmentation step a SACSegmentation script, that receives the filtered Point Cloud and returns the planes which the cloud contains, was used.

After segmenting the points into planes, a 48×48 matrix where each cell represents a square of $25cm^2$,is used. The size of the matrix was defined based on an assumption that the robot can accurately see $10m$ ahead and $6m$ to each side. In the future we also plan on saving $2m$ of past data, in order to have the ability to safely retrace our steps, if needed. The size of each square was defined by estimating the size of the foot of the DRC robot, thus allowing or disallowing placing of a foot in any specific square. This module is defined as a service, and therefore data will be provided to any client upon request via ROS' publish and subscribe methods.

By using the localization module we can map the terrain from a fixed axis, this lowers the need computation in reading the map and provide a better base to detect inconsistency in the parsed data

Result

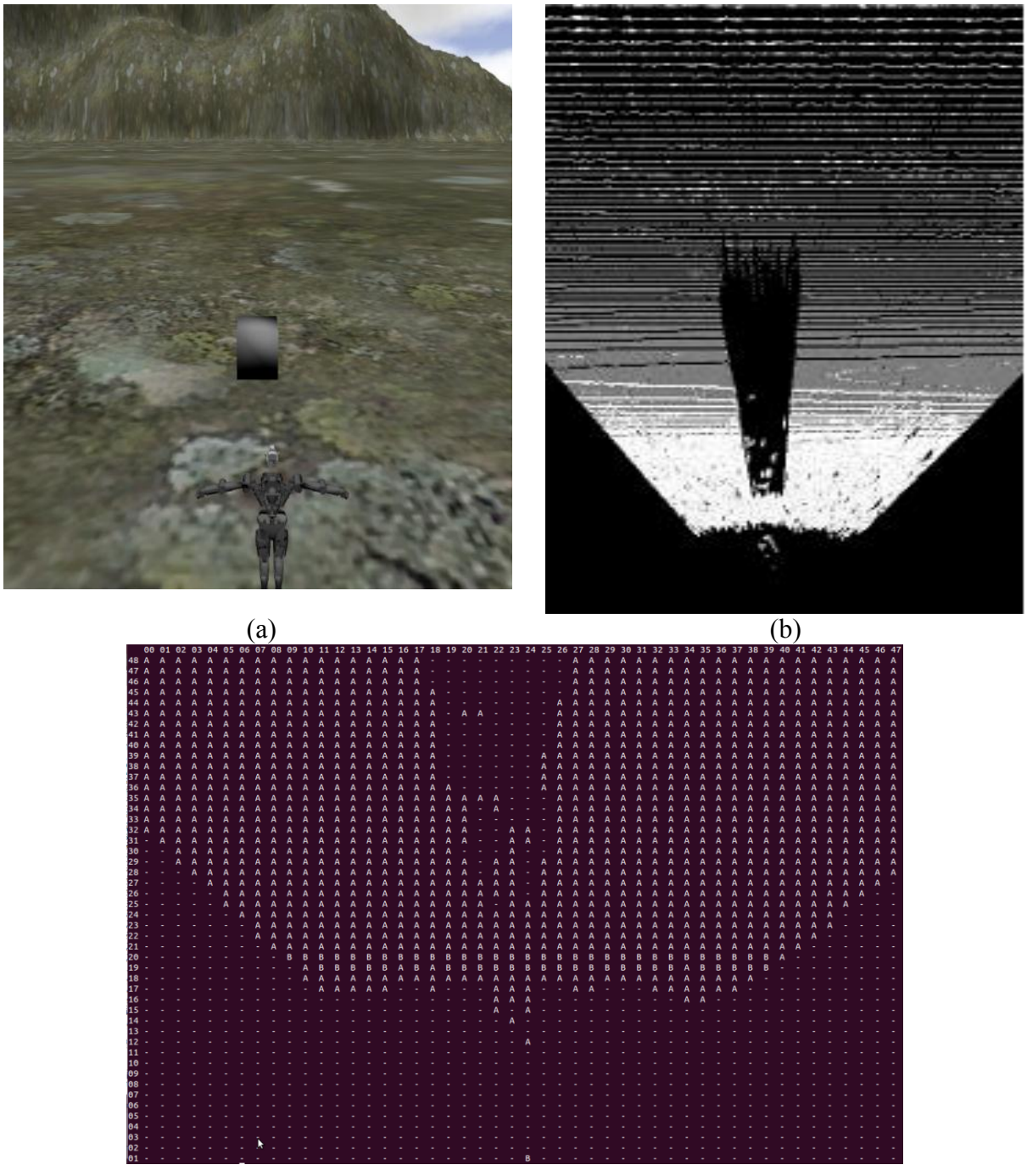
After implementing the code so it calls upon all the required ROS packages, a ROS service was created. The service can transform a snapshot taken from the DRC robot's camera into an eagle-eye map of the terrain on demand. Two examples can be seen in the following Figures.



(a) (b)

(c)

FIGURE 3.2.2: (a) DRC robot in an empty world with texture, (b) RVIZ of the image, (c) eagle-eye representation



(a) (b)

(c)

FIGURE 3.2.3: (a) DRC robot in an empty world with texture and obstacle, (b) RVIZ of the image, (c) eagle-eye representation

2.2.3 Module C23 – Object Recognition

Introduction

One of the main tasks of the robot is to successfully detect surrounding objects. For example, in order to successfully drive the car, first the robot should detect the car, reach the car and detect the steering wheel. The main purpose of this module is to detect all the tools that the

robot will interact with, such as cars, wheels, doors, water plugs etc…
This module can search for multiple objects, where for each object, the robot module will return a Boolean number (1|0) indicating whether it was detected or not.
If the object is detected, the module returns a bounding box around the object.

Method
For this to work, the module will implement two algorithms:
Classifier/ Detector: Used for the recognition phase. The module will assign a separate classifier per object.
Tracker: After the object was detected, the module will track the trajectory of the object in the next frames, it will locate the object and update whether the robot is on the right track or not.
Object Server: key object's and their location are saved in the object server for future reference and mapping,

Implementation
The TLD algorithm, (Track, Learn and Detect) which is a real-time algorithm for tracking of previously unseen objects in video streams, is employed. The object of interest is defined by a bounding box in the first frame. The result is real-time tracking of the object that improves over time.
The algorithm was trained using the different orientations and scales of the door. This way algorithm learns how the object looks like, so that with time the object tracking and detecting improves. The object server is still under development.

2.2.4 Module C24 – Obstacle Detection

Introduction
This module detects obstacles both under ground level (holes) and above ground level, including information on data confidence provided for the position of the detected obstacle and also provides an eagle-eye view of the locations of the obstacles detected.

Method
Using entropy based segmentation; deferent areas on a given camera data are labeled, areas not detected as ground are immediately marked as obstacles until further confirmation is given. While driving the C51 node is using the Lidar data, for fast detection of height changes a head of its path.

2.2.5 Module C25 – Global Position

Introduction
This module provides the robot's position in the world using visual odometery and IMU data.

Method
In order to calculate the robot's location, a visual odometry node is used to provide estimation of the robot position and accompany it with data from the IMU for error deduction.

Implementation
Using the LibViso2 position estimation, the odometry and IMU data are passed to a Kalman filter to provide an estimation of the robot movement.

2.3 Decision Making

We implemented the ability to view at real time execution debug information (green denotes that the task is being executed) and history (Figure 3.3.1, each task can be queried about past values).

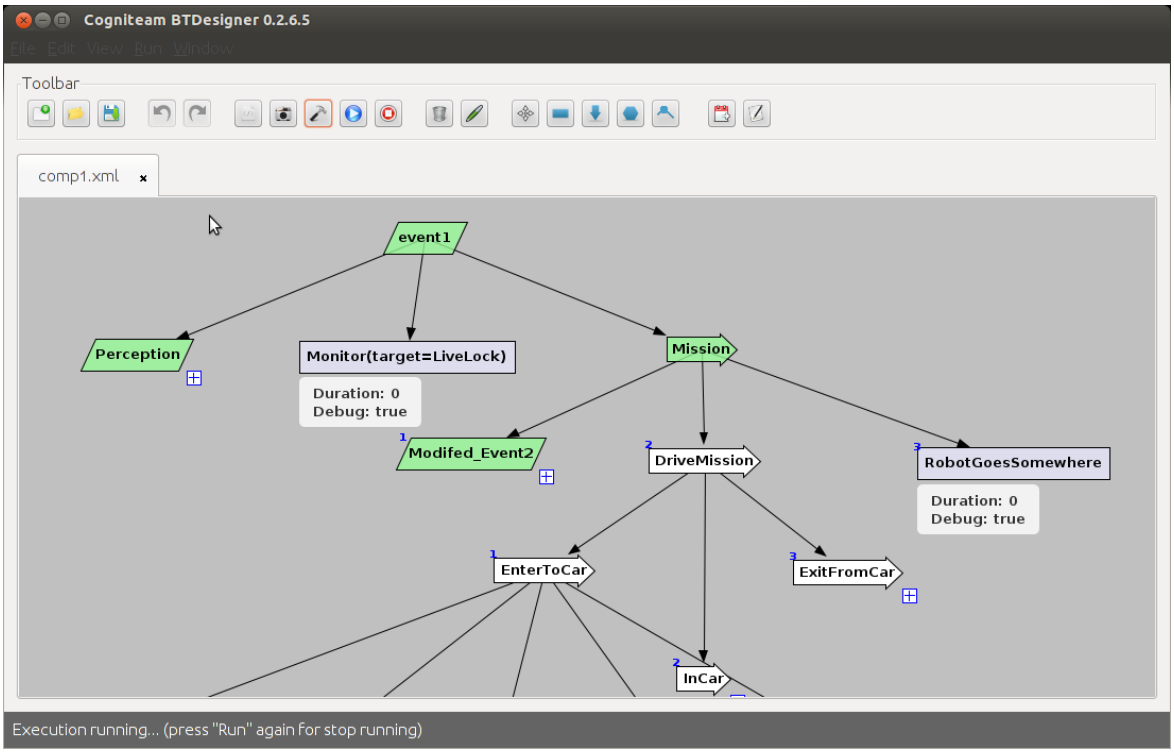


FIGURE 3.3.1: ROBIL DESIGNER WITH RUNTIME INFORMATION

Description of the task is now part of the workspace environment allowing maintenance and update of task description both the programmer and the user (Figure 3.3.2).

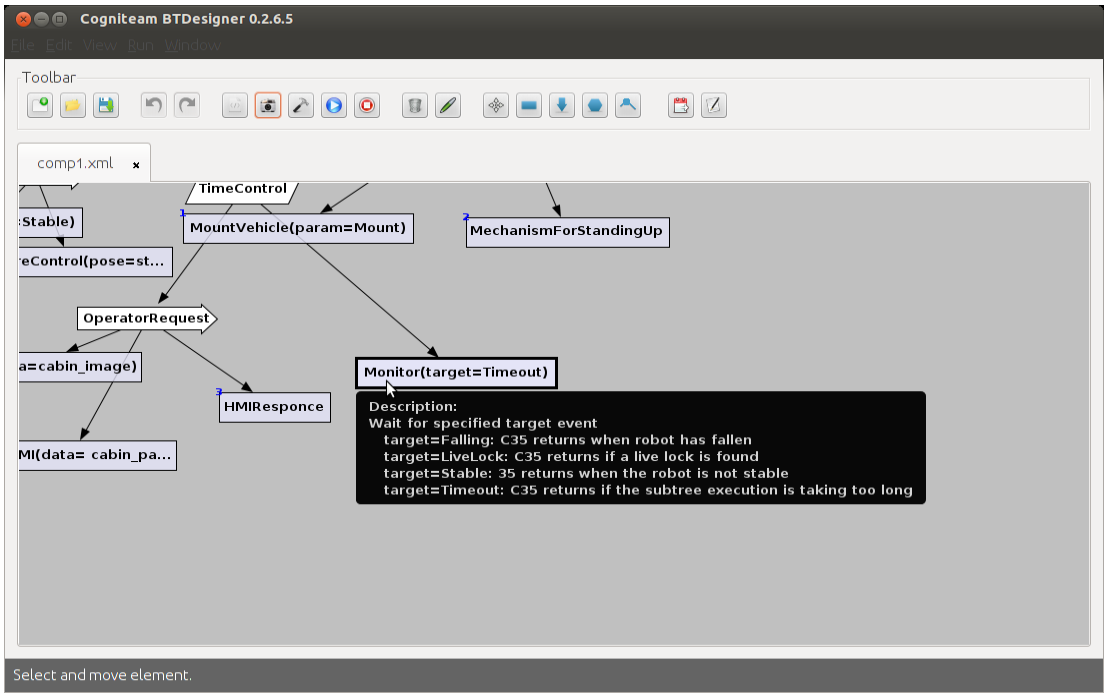


FIGURE 3.3.2: ROBIL DESIGNER TASK DOCUMENTATION

Mission and sub trees are now collapsible and can be exported to create and save new sub-trees [1-6] (Figure 3.3.3)

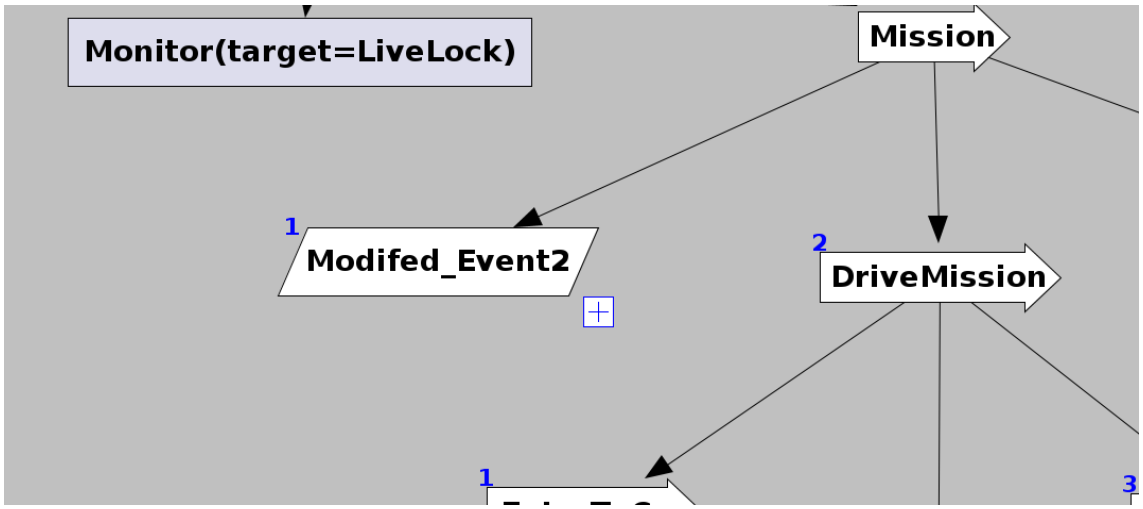


FIGURE 3.3.3: ROBIL DESIGNER BEHAVIOUR TREE FOLDING

Regarding task execution in order to allow fast execution of concurrent tasks we modified the action lib clients and servers to better cope with concurrent goals (both for python and C++).

We also have made progress with our monitoring module

1. For monitoring tasks were implemented: Live lock, Stability, Falling, and Timeout:

    1. Falling returns when robot has fallen. Based on information passed on topic /C35/falling

2. Live lock: returns if the monitored node is taking longer than 1.X times slower than expected, expected time is retrieved from each module.
3. Stable: returns values [1-5] corresponding to the robot's stability. Based on information passed on topic /C35/stability
4. Timeout: returns if the subtree execution is taking longer than X, where X is given as a parameter [6-9].

2. We support task-specific monitoring. These are monitoring tasks that modules want to maintain externally. As an example we implemented in Waypoint Monitor which monitors the avg. distance between waypoints. If this average distance is above some threshold, an alert is issued.

3. We implemented progress monitoring to be used by the HMI. Based on a comparison of the progress made online with the expected progress for that time point and the deadlines for task completions, it gives recommendations for speeding up execution.

Our current focus is mainly addressed to debugging and integration for all the behavior trees built by the different teams. The future plans include:

1. Statistics and data collection on the execution times of modules.
2. We implemented an early version of a remote Designer with communication optimized to the competition rules, allowing following execution remotely while using network in a smart manner. Challenges arose from ROS underling communication infrastructure, which is not optimized to deal with communication constraints. This module needs further elaboration and connection to the HMI.
3. We plan to implement executional condition specification (conditions that relate to the execution trace of the mission), and the construction of behavior trees at real time, by the use of NLP driven methods.

2.4 Dismounted Mobility

Dismounted mobility was led and developed by the Technion group, including vehicle mounting.

2.4.1 Locomotion

In order to complete Task 2 of the Virtual Robotics Challenge we originally intended to extend our successful CPG-based, dynamic walking controller, as well as a ZMP implementation as a fallback [10-12]. However, due to the tight time frame of the competition and the ever changing simulation framework we decided to take advantage of the Boston Dynamics Interface (BDI) controller supplied with the DRC environment, for biped walking. In order to cross the mud-pit we developed a robust pentad mode of walking, which was also extended to crossing the hills and debris area.

Thus, we have developed four walking modes, to facilitate crossing different types of terrain. Two modes, the continuous dynamic walk and the pentad walk **allowed successful completion of task-2 as demonstrated in the video and in subsequent Figures.**

1. Continuous-dynamic walk (CDW): supports biped locomotion to the target, or a sequence of via-points. It automatically generates positions for foot-placement and

sends the required foot placement sequence to the Boston Dynamics dynamic walking module. It is adequate for walking in the pen, walking on flat terrain and crossing the debris area. In the actual competition it was used to exit the pen and to get to the vicinity of the car or table.

2. Discrete-foot-placement walk (DFPW): supports biped locomotion, either quasi-statically or dynamically, based on an externally specified vector of foot placements. It could be used to cross the hills quasi-statically, as shown in Figure 3.4.1, provided that proper foot-placements are specified [13-16]. It was not used in the actual competition.

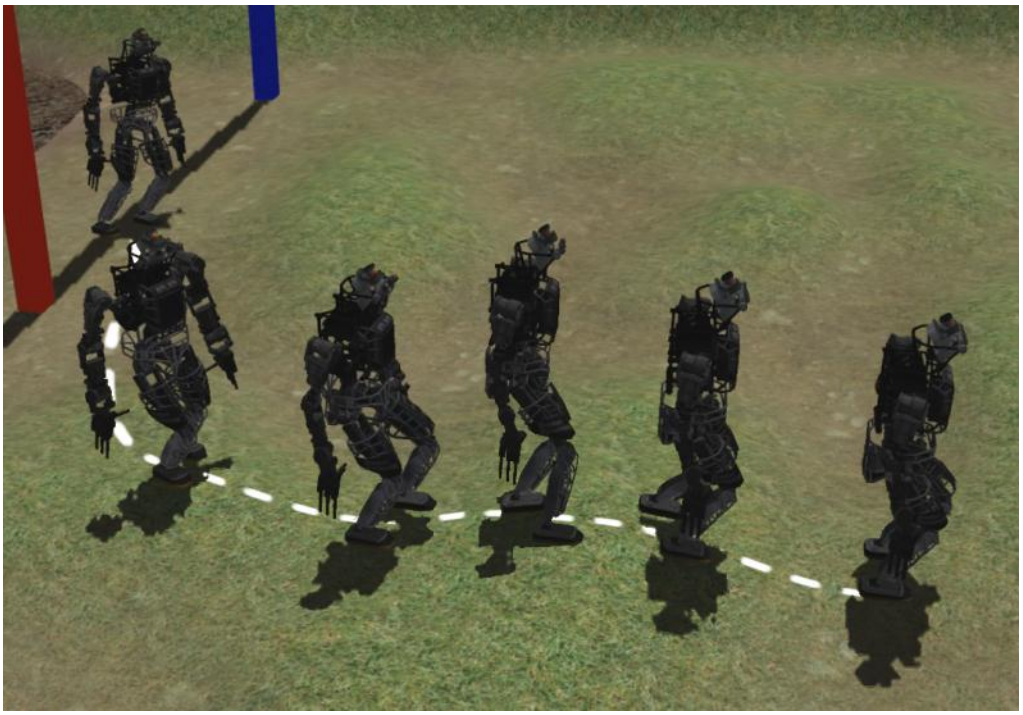


FIGURE 3.4.1: Hill crossing using dynamic foot-placement walk (DFPW). The robot is able to walk slowly by stepping on specific locations were the robot can remain stable.

3. Pentad-walk (PW): We developed a pentad gait framework that included a safe sit-down sequence, forward pentad-walking to enter the mud pit and backwards pentad-walking to exit the mud pit, as depicted in Figure 3.4.2, and in the above movie. Furthermore, two different motion sequences were designed to turn in place under low and normal ground friction conditions.

   The PW uses both hands and feet to support the weight of the robot, facing up. The pelvis is then lifted, moved forward/backwards and placed back on the ground. This is followed by a forward/backward motion of the legs and hands. This locomotion method is more stable than the bipedal methods since the COM is closer to the ground and the support polygon is much larger. The basic mode can support motion perpendicular to the terrain's contour lines, and was developed for entering, crossing and exiting the mud-pit.

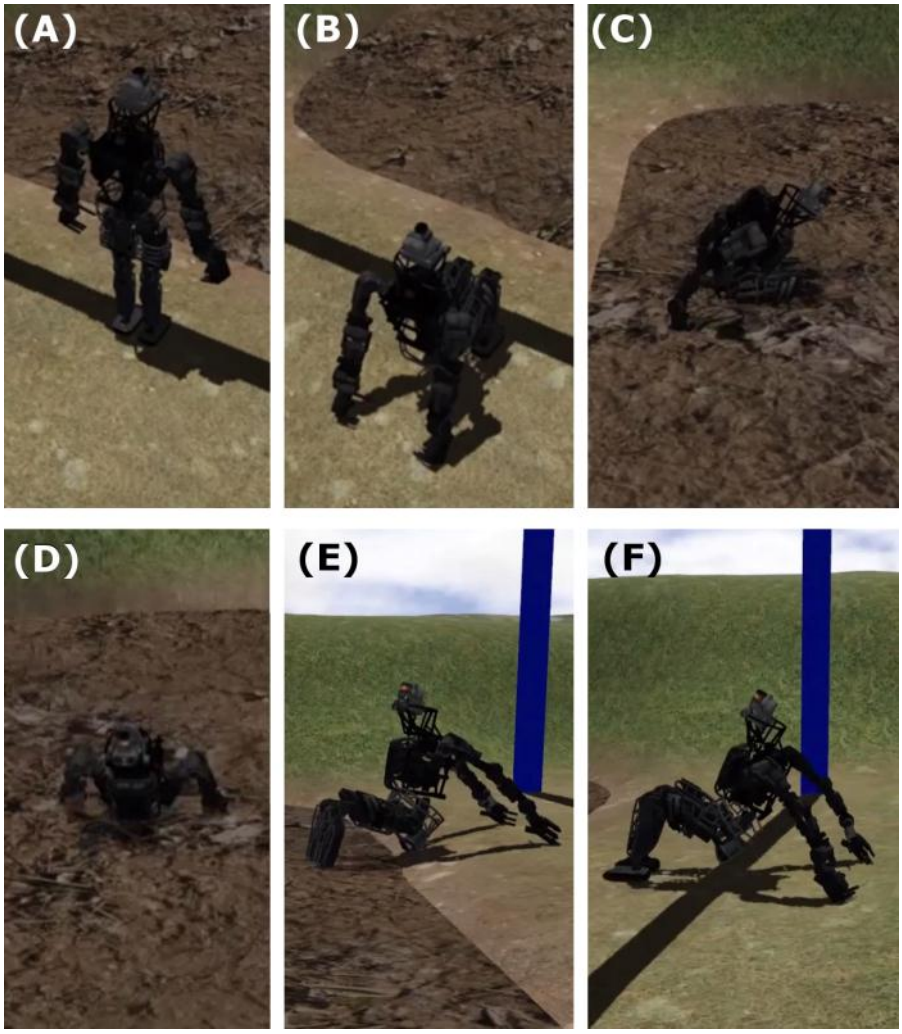


FIGURE 3.4.2:Mud-pit crossing using pentad walk (PW), see movie**:** (A) Approach mud, (B) sit down, (C) enter mud walking forward, (D) turn 180 degrees inside mud pit, (E) exit mud walking backwards, (F) cross mud gate

Our PW framework was also very successful at crossing the hills section and debris section of the task, especially by moving backwards, as shown in Figure 3.4.3, and 3.4.4, respectively, and in the above movie. Crossing the hills and the debris area presented a new challenge since it may cause the robot to fall. Hence we enhanced the basic algorithm in two ways: (i) feedback and compliance were added to minimize falling, and (ii) an algorithm for fall recovery was developed. Special motion sequences were tailored to recover from left, right, forward and backwards tipping [8-10]. A higher level controller was designed to autonomously walk towards points along a path, correcting for orientation drift and recovering from falls. The robot can autonomously detect a failure to advance and locally adjust the path to circumvent the problematic area.

Using our pentad-walking framework ATLAS is able to cross from gate to gate autonomously, using only a 2-point path, even with a noisy odometry, as long as the cumulative errors are smaller than half the gate's width. In the actual competition

pentad-walk was used successfully to cross the mud and the hills and to start crossing the debris, as shown in the following clip

FIGURE 3.4.3: Hill crossing using pentad-walk (PW, see movie).
The robot crosses the hill section autonomously, walking towards the center of the next gate. The robot is able to recover from tipping in any direction and can locally modify its path if it gets stuck [16-21].

FIGURE 3.4.4: Debris crossing using pentad-walk (PW, see movie). The robot crosses the hill section autonomously, walking towards the center of the next gate. The robot is able to recover from tipping in any direction and can locally modify its path if it gets stuck.

4. Translation: supports continuous dynamic translation relative to the current robot position. This mode was developed to facilitate getting near the car or table and to allow making short steps.

In addition the dismounted mobility included capabilities to:

5. Sit down and stand up to support transition from biped to pentad locomotion.

6. Turning in either pentad or biped. Turning in pentad-state was used to turn in the middle of the pit to allow the robot to exit stably by walking backward.

2.4.2 Vehicle Mounting and Dismounting

Our first approach for getting into the car involved grasping the car's frame and then stepping into the car with the right leg. This approach wasn't robust and while the robot was able to reach the seat's height, we were unable to get it to enter the car.

Since we lacked autonomous perception and grasping capabilities to perform the task in this manner, we decided to go with an open-loop motion, generated manually. We have attempted motions that jump in and out of the vehicle, fall in or out, enter/exit from the sides by stepping or sliding, or crawling. Finally, thinking outside box, wrap the arms around the car's frame (on the passenger side), lift the robot's legs and swing into the car, as depicted in Figure 3.4.5, and demonstrated in the video. We ensured that the robot laid flat on the seat while getting the arms inside the car and then performed a sit-up motion that ends with the robot facing the steering wheel. Due to friction issues between ATLAS and the car we added some arm and leg motions that greatly improve the repeatability of the motion sequence. We added an optional motion to press the gas pedal at the end of the sequence in order to secure additional points for crossing the first gate with the car. However, getting into the car, the robot would randomly presses the reverse button causing unexpected results.

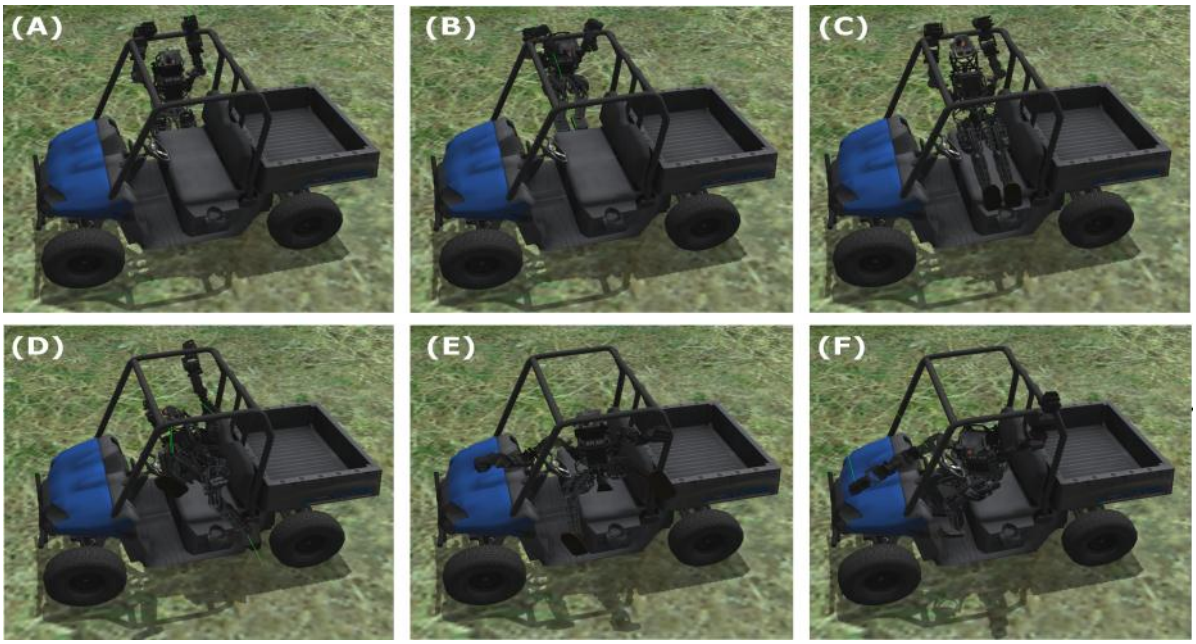


FIGURE 3.4.5: Swing into car motion sequence. (A) Wrap arms around frame, (B) lift legs, (C) swing into car, (D) release frame to fall on seat, (E) sit-up and (F) rotate to face the steering-wheel.

3.4.3 Software Design

The intent of the software design chosen for the Dismounted Mobility module was to allow for several different walking modes to be interchangeable while keeping the same behavioral mechanism through which the module interacts with other modules.

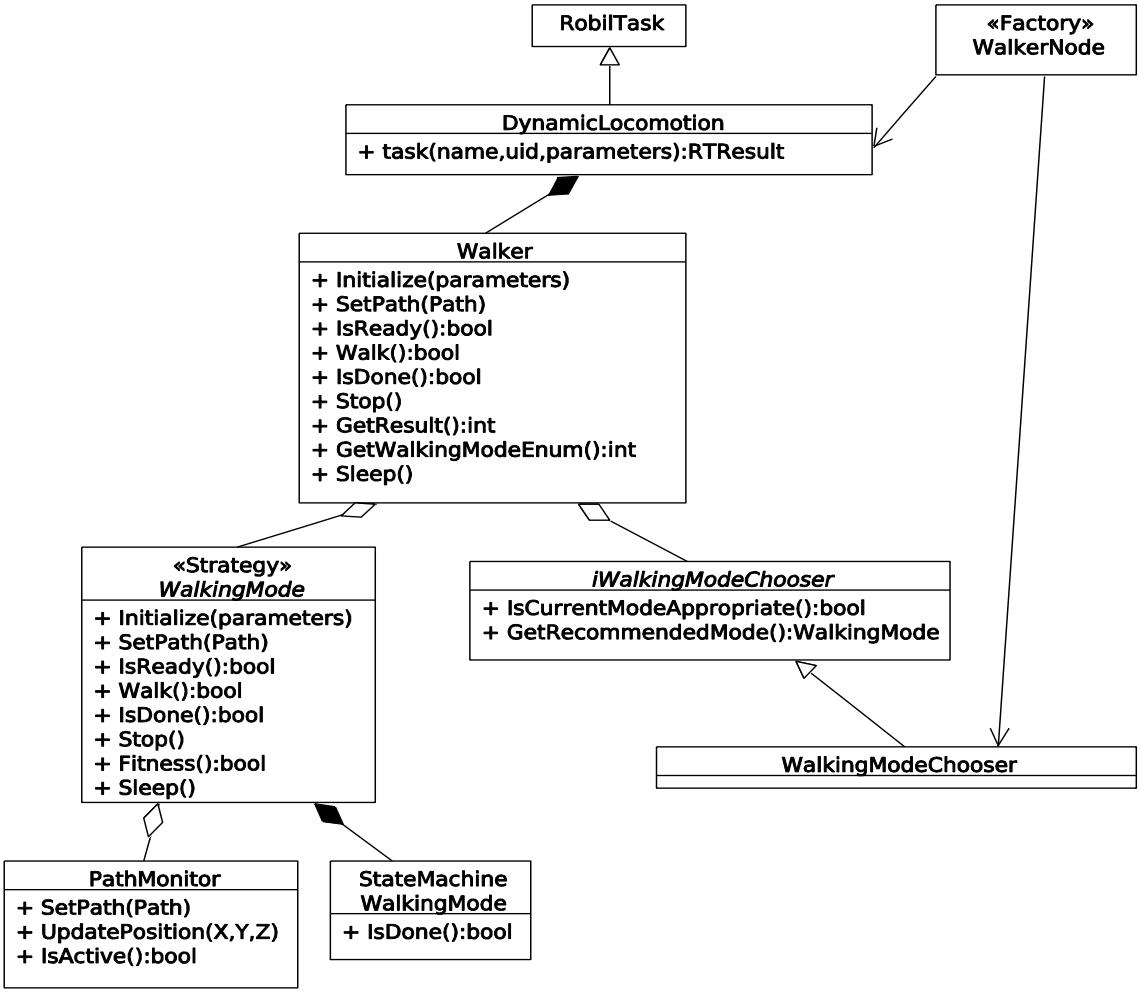


FIGURE 3.4.6: Dynamic Locomotion class diagram for Dismounted Mobility

As seen in the class diagram, Figure 3.4.6, a class called *DynamicLocomotion* is a *RobilTask* which, when a task is provoked, runs a predefined sequence consisting of getting ready to walk, walking, stopping and sending out the result of the walking task. The *DynamicLocomotion* has a *Walker* class instance, which encapsulates the *WalkingMode* and *WalkingModeChooser*classes. The *Walker* is responsible for the correct performance of the walking sequence by the right *WalkingMode*. If the *WalkingMode* isn't fit to perform, the *WalkingModeChooser* is used to replace the inadequate *WalkingMode* with a more appropriate one.

An example for a need to replace one *WalkingMode* with another happens when a Discrete-foot-placement path has changing heights in it, which requires a change from dynamic walking mode to quasi-static, without interrupting the task sequence.

In order to sustain a uniform behavioral mechanism independent of actual walking modes, while allowing higher levels of decision making to have control over the actual walking modes used, a *Factory* design pattern was introduced in the form of the *WalkerNode*. The *WalkerNode* is responsible for the construction of the *DynamicLocomotion* class and the construction of a concrete *WalkingModeChooser* to accompany the *DynamicLocomotion*. The higher level decision making module can thus control which *WalkingMode* instances are available for the *WalkingModeChooser* and accordingly override the control over which walking mode is used during the performance of the task.

The *WalkingMode* is an abstract class which is responsible for the actual walking to take place, as shown for example in Figure 3.4.7. It has a *PathMonitor* to manage the progress over a given path and a *WalkingModeStateMachine* which manages the different phases of the walking task

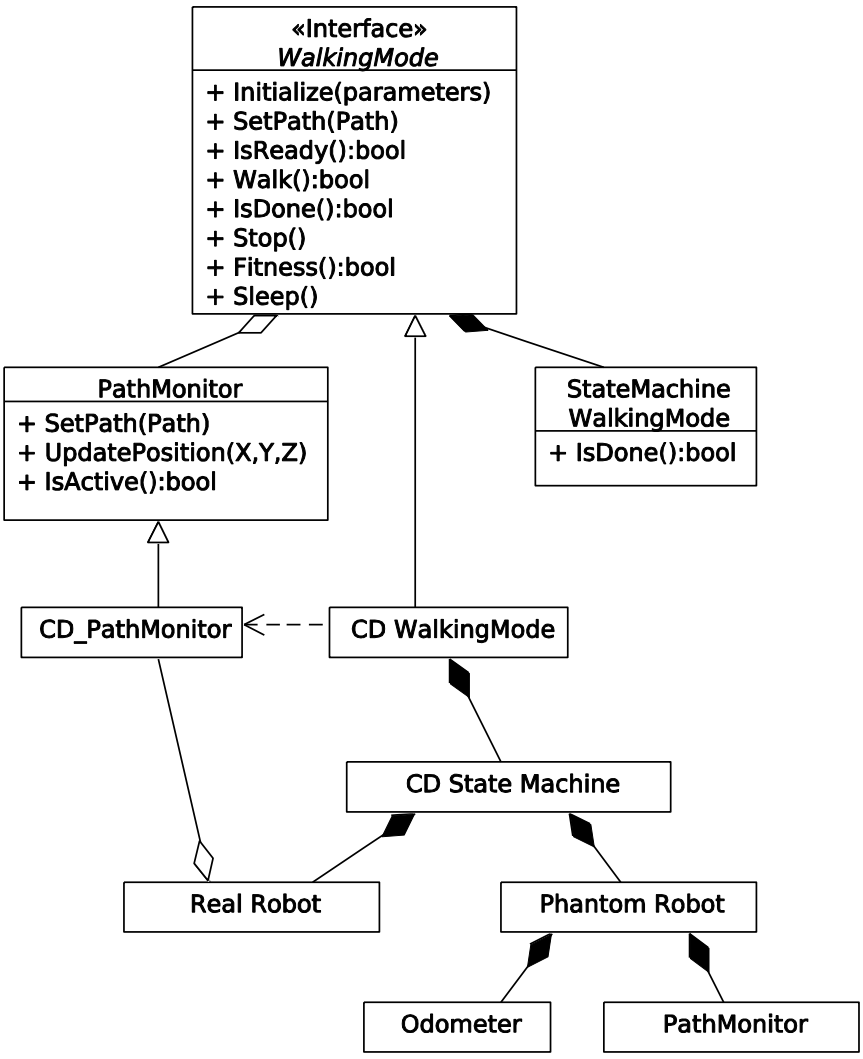


FIGURE 3.4.7: An actual Walking Mode and its inheritance relationship with the abstract Walking Mode class

In the class diagram shown in Figure 3.4.7, the *Continuous Dynamic Walking Mode* (CDWM) class architecture is shown as an example of how the strategy design pattern was used. Using this pattern, all walking modes were easily fitted into the general design. The *WalkingModeChooser* class is an aggregation class designed to hold a group of *WalkingMode* strategy instances and choose among them which is best suited for the upcoming path.

As shown in Figure 3.4.8, the *Walker* class has a *WalkingModeChooser* which in turn has a function that returns a boolean answer to the question "Is the current walking mode appropriate?". The *WalkingMode* class has a *Fitness()* function which returns true if the *WalkingMode* is fit to continue or false otherwise. The *WalkingModeChooser* may have several Walking Modes, and a key for knowing how desirable each Walking Mode is (set by

higher levels through the *WalkerNode* factory). If the current mode is unfit, or if there is a more desirable mode which is fit, the *WalkingModeChooser* indicates that the current mode is inappropriate and recommends on a different walking mode instead.

Through the use of the strategy pattern, the Walker may change from one walking mode to another without any awareness as to which mode is which, the whole mechanism is facilitated by the *WalkingModeChooser* indicating that a change is needed, and supplying the Walker class with a replacement *WalkingMode*.

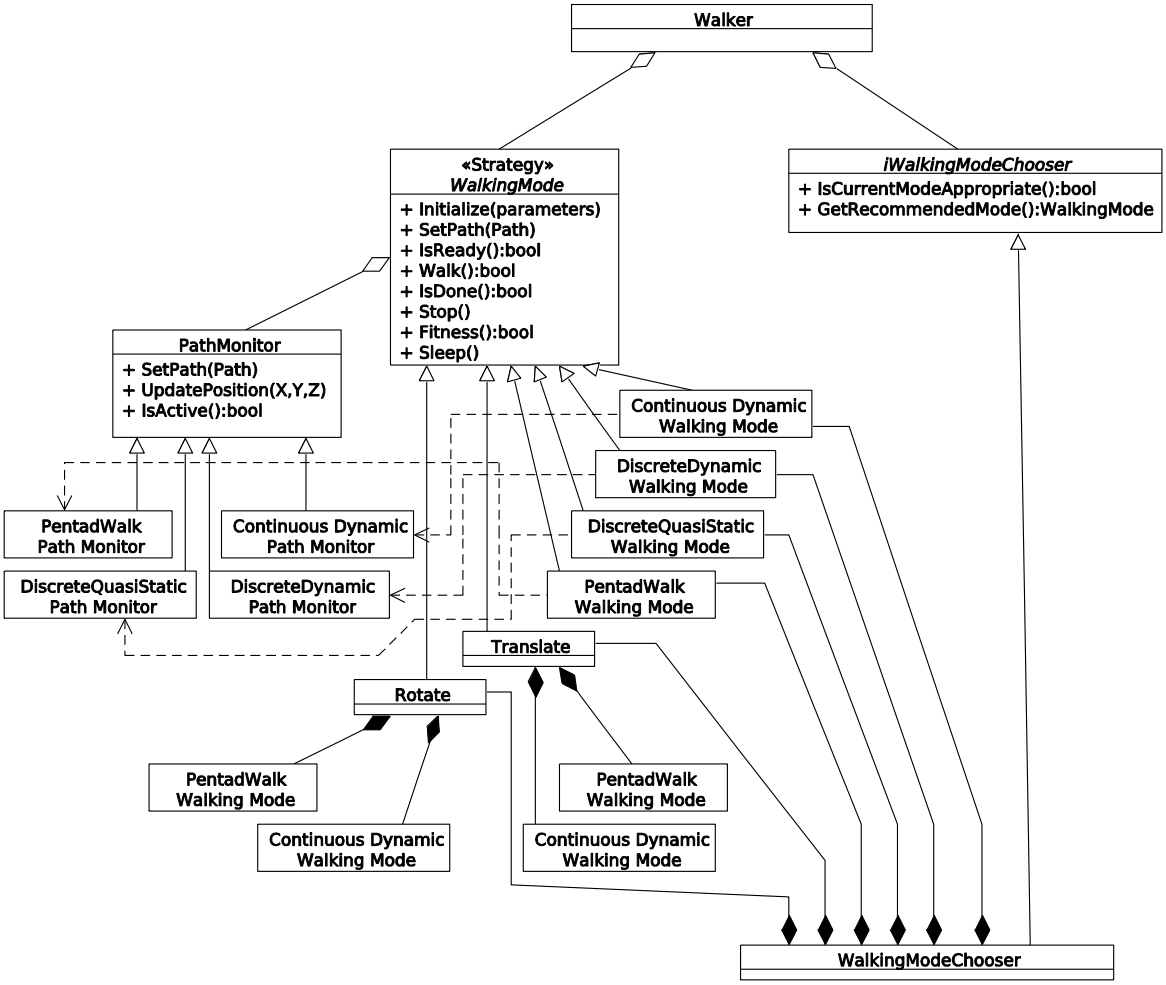


FIGURE 3.4.8: The different Walking Modes available to the Walking Mode Chooser and their relationships

2.5 Mounted Mobility

2.5.1 Introduction

One of the tasks anointed by DARPA is driving a utility vehicle. The utility vehicle is non-holonomic, i.e. the turning abilities of the vehicle is limited and the vehicle can't turn in place.

In order to drive the non-holonomic vehicle, the robot will build a path, and execute the driving process based on the chosen trajectory [22, 23]. Thus, we will distinguish between

two (2) different processes related to this task. The first, the robot will create a trajectory on the fly. This procedure will be executed by "Decision Making", in module C31 - "Path Planning". The output of C31 is a series of way points that will be distributed non-homogeneously along the trajectory, according to path condition. The second process is to drive the car through the way points given by C31. The car is driven using a fuzzy logic controller (FLC), alongside a reactive controller that will aid in the obstacle avoidance process.

In this chapter we will describe the fuzzy logic controller, communication with the different modules and show the results achieved.

2.5.2 Assumptions

We assume that Atlas is situated in the vehicle, in an upright position in front of the steering wheel and in a static position. One of our main issues was keeping Atlas situated in the seat. Due to the low friction of the seat, Atlas always slipped and fell off the seat.

Another assumption is that all of the modules are running and are integrated with each other, i.e., decision making is integrated with perception and the global position modules; dexterity is integrated with the object recognition. We also assume that the global position module (x_{car}, y_{car}) and the IMU (θ_{car}) are operating with reasonable error between way points.

2.5.3 Methods

Fuzzy Logic Controller

We will create a negative feedback control loop using the location (x_d, y_d) of the way point we wish to arrive at, and the location, orientation and orientation rate of the car $(x_{car}, y_{car}, \theta_{car}, \dot{\theta}_{car})$, as described in 0, throughout the simulation.

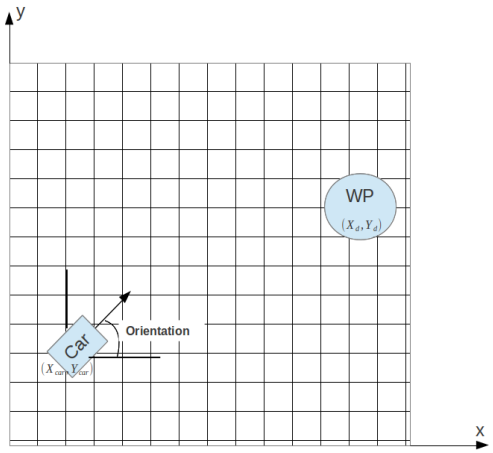


FIGURE 3.5.1: Car and Way Point sketch

With these five (5) variables $(x_d, y_d, x_{car}, y_{car}, \theta_{car})$ we can calculate two (2) new variables: Distance from the car to way point (d_e). The distance is defined as follows:

$$d_e = \sqrt{(x_d - x_{car})^2 + (y_d - y_{car})^2} \quad (3.5.1)$$

Orientation error (θ_e) - Defined as the difference between the orientation of the car and the slope angle of the car and the way point. The orientation error is calculated as follows:

$$\theta_e = \tan^{-1}\frac{y_d - y_{car}}{x_d - x_{car}} - \theta_{car} \qquad (3.5.2)$$

These two (2) new variables along with the orientation rate ($\dot{\theta}_{car}$) will be the input variables of the fuzzy logic controller (FLC) in order to control the vehicle. The output of the FLC will be the desired velocity and the desired angle of the steering wheel [24-28].

Fuzzy Logic Membership functions

In order to turn the crisp inputs into fuzzy variables, we will define the following normalized (i.e. values between 0 and 1), membership functions:

2.5.3.1 Orientation error membership function

For the orientation error membership function, we will assign five (5) linguistic variables: Big Negative (BN); Small Negative (SN); Zero (Z); Small Positive (SP); & Big Positive (BP). Let us denote the crisp input of the orientation error as θ_e, and define the mathematical relation of the linguistic variables with θ_e as follows:

$$BN(\theta_e) = \begin{cases} 1 & -1 < \theta_e < -\frac{60}{180} \\ \frac{-30 - 180\cdot\theta_e}{-30-(-60)} & -\frac{60}{180} < \theta_e < -\frac{30}{180} \\ 0 & -\frac{30}{180} < \theta_e \end{cases} \qquad SN(\theta_e) = \begin{cases} 0 & \theta_e < -\frac{60}{180} \\ \frac{180\cdot\theta_e - (-30)}{-30-(-60)} & -\frac{60}{180} < \theta_e < -\frac{30}{180} \\ \frac{0 - 180\cdot\theta_e}{0-(-30)} & -\frac{30}{180} < \theta_e < 0 \\ 0 & 0 < \theta_e \end{cases}$$

$$Z(\theta_e) = \begin{cases} 0 & \theta_e < -\frac{10}{180} \\ \frac{180\cdot\theta_e - (-10)}{0-(-10)} & -\frac{10}{180} < \theta_e < 0 \\ \frac{10 - 180\cdot\theta_e}{10-0} & 0 < \theta_e < \frac{10}{180} \\ 0 & \frac{10}{180} < \theta_e \end{cases}$$

$$SP(\theta_e) = \begin{cases} 0 & \theta_e < 0 \\ \frac{180\cdot\theta_e - 0}{30-0} & 0 < \theta_e < \frac{30}{180} \\ \frac{60 - 180\cdot\theta_e}{60-30} & \frac{30}{180} < \theta_e < \frac{60}{180} \\ 0 & \frac{60}{180} < \theta_e \end{cases} \qquad BP(\theta_e) = \begin{cases} 0 & \theta_e < \frac{30}{180} \\ \frac{180\cdot\theta_e - 30}{60-30} & \frac{30}{180} < \theta_e < \frac{60}{180} \\ 1 & \frac{60}{180} < \theta_e < 1 \end{cases}$$

The corresponding membership function schematic is described in 0.

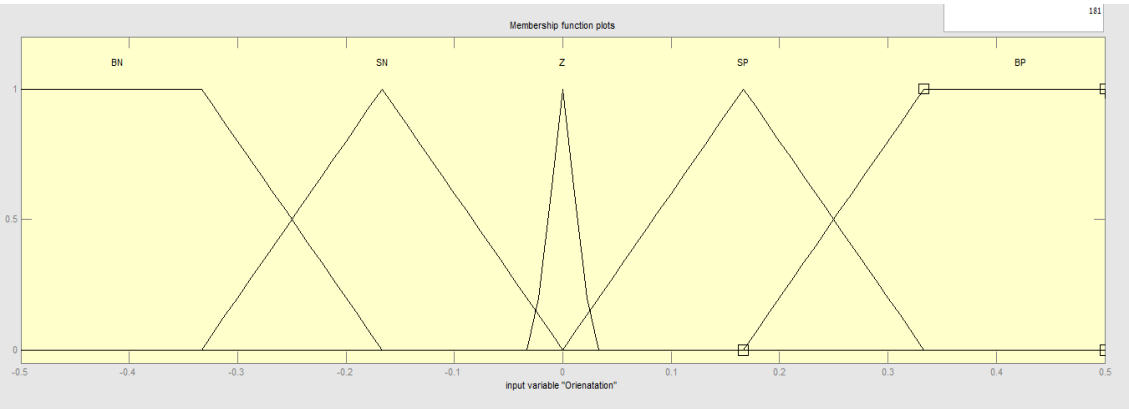


FIGURE 3.5.2: MATLAB orienation error membership function schematic.

2.5.3.2 Change in orientation error membership function

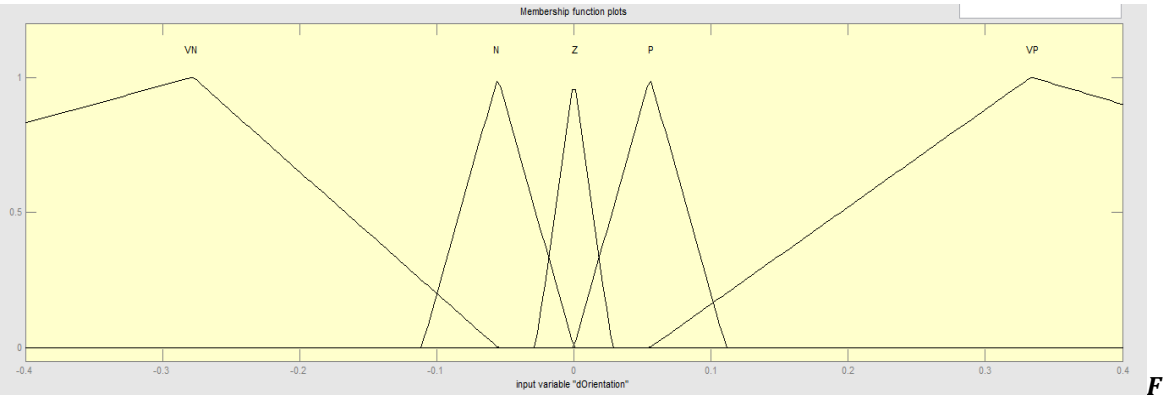
For the orientation error membership function, we will assign five (5) linguistic variables: Very Negative (N); Negative (N); Zero (Z); Positive (P); & Very Positive (P). Let us denote the crisp input of the change in orientation error as $\dot{\theta}_e$, and define the mathematical relation of the linguistic variables with $\dot{\theta}_e$ as follows:

$$VN(\dot{\theta}_e) = \begin{cases} 1 & -1 < \dot{\theta}_e < -\frac{60}{180} \\ \frac{-30 - 180 \cdot \dot{\theta}_e}{-30 - (-60)} & -\frac{60}{180} < \dot{\theta}_e < -\frac{30}{180} \\ 0 & -\frac{30}{180} < \dot{\theta}_e \end{cases} \qquad N(\dot{\theta}_e) = \begin{cases} 0 & \dot{\theta}_e < -\frac{60}{180} \\ \frac{180 \cdot \dot{\theta}_e - (-30)}{-30 - (-60)} & -\frac{60}{180} < \dot{\theta}_e < -\frac{30}{180} \\ \frac{0 - 180 \cdot \dot{\theta}_e}{0 - (-30)} & -\frac{30}{180} < \dot{\theta}_e < 0 \\ 0 & 0 < \dot{\theta}_e \end{cases}$$

$$Z(\dot{\theta}_e) = \begin{cases} 0 & \dot{\theta}_e < -\frac{10}{180} \\ \frac{180 \cdot \dot{\theta}_e - (-10)}{0 - (-10)} & -\frac{10}{180} < \dot{\theta}_e < 0 \\ \frac{10 - 180 \cdot \dot{\theta}_e}{10 - 0} & 0 < \dot{\theta}_e < \frac{10}{180} \\ 0 & \frac{10}{180} < \dot{\theta}_e \end{cases}$$

$$P(\dot{\theta}_e) = \begin{cases} 0 & \dot{\theta}_e < 0 \\ \frac{180 \cdot \dot{\theta}_e - 0}{30 - 0} & 0 < \dot{\theta}_e < \frac{30}{180} \\ \frac{60 - 180 \cdot \dot{\theta}_e}{60 - 30} & \frac{30}{180} < \dot{\theta}_e < \frac{60}{180} \\ 0 & \frac{60}{180} < \dot{\theta}_e \end{cases} \qquad VP(\dot{\theta}_e) = \begin{cases} 0 & \dot{\theta}_e < \frac{30}{180} \\ \frac{180 \cdot \dot{\theta}_e - 30}{60 - 30} & \frac{30}{180} < \dot{\theta}_e < \frac{60}{180} \\ 1 & \frac{60}{180} < \dot{\theta}_e < 1 \end{cases}$$

The corresponding membership function schematic is described in 0.



F

FIGURE 3.5.3: MATLAB 'change in Orientation error' membership function schematic.

2.5.3.3 Distance membership function

For the distance membership function, we will assign three (3) linguistic variables: Close, Far, and very Far. Let us denote the crisp input of the distance as d_e, and define the mathematical relation of the linguistic variables with d_e as follows:

$$Close(d_e) = \begin{cases} 1 & 0 < d_e < 0.05 \\ \frac{0.15 - d_e}{0.15 - 0.05} & 0.05 < d_e < 0.15 \\ 0 & 0.15 < d_e \end{cases} \qquad Far(d_e) = \begin{cases} 0 & d_e < 0.1 \\ \frac{d_{e-0.1}}{0.2 - 0.1} & 0.1 < d_e < 0.2 \\ \frac{0.2 - d_e}{0.2 - 0.1} & 0.2 < d_e < 0.4 \\ 0 & 0.4 < d_e \end{cases} \qquad Very\ Far(d_e) = \begin{cases} 0 & 0.2 < d_e \\ \frac{d_e - 0.2}{0.4 - 0.2} & 0.2 < d_e < 0.4 \\ 1 & 0.4 < d_e < 1 \end{cases}$$

The corresponding membership function schematic is described in 0.

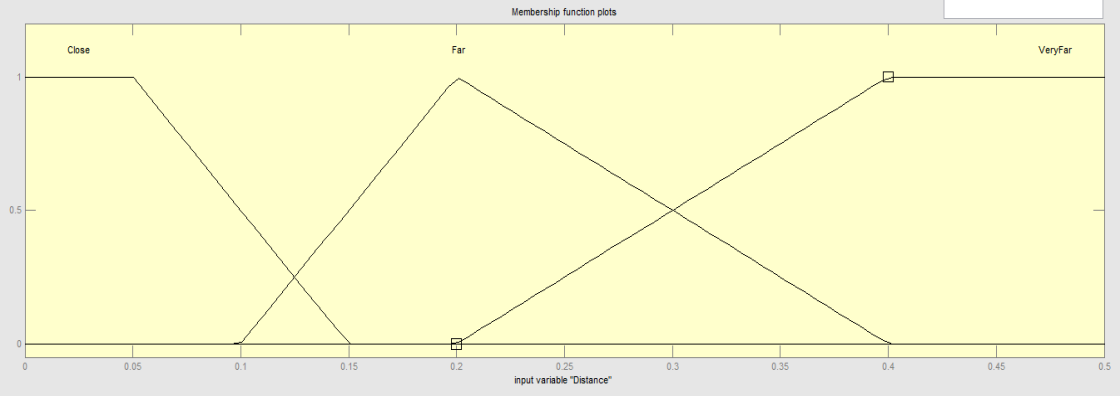


FIGURE 3.5.4: MATLAB 'Distance' membership function schematic.

2.5.3.4 Steering angle membership function

Before we assign the membership function, let us note that we will grip the steering wheel with one hand in order to avoid any complications with changing hands while driving. For this matter we will grip the steering wheel from the highest point and will denote this angle as 0[rad]. We will now turn the steering wheel left and right up too $\frac{\pi}{2}$[rad] to each direction. This will make the turning process simple and effective as it answers for most of the turning operations required.

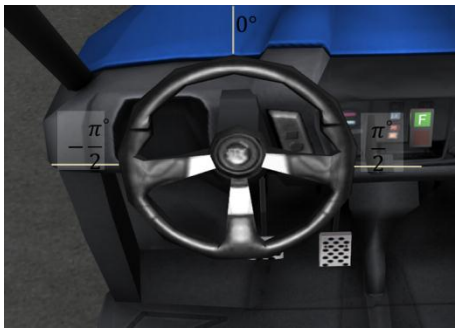


FIGURE 3.5.5: Hand wheel grip at angle 0.

For the steering wheel angle membership function, we will assign also five (5) linguistic variables: Hard Left (HL); Soft Left (L); Straight (ST); Soft Right (R); & Hard Right (HR). The steering wheel angle membership function schematic is shown in 0.

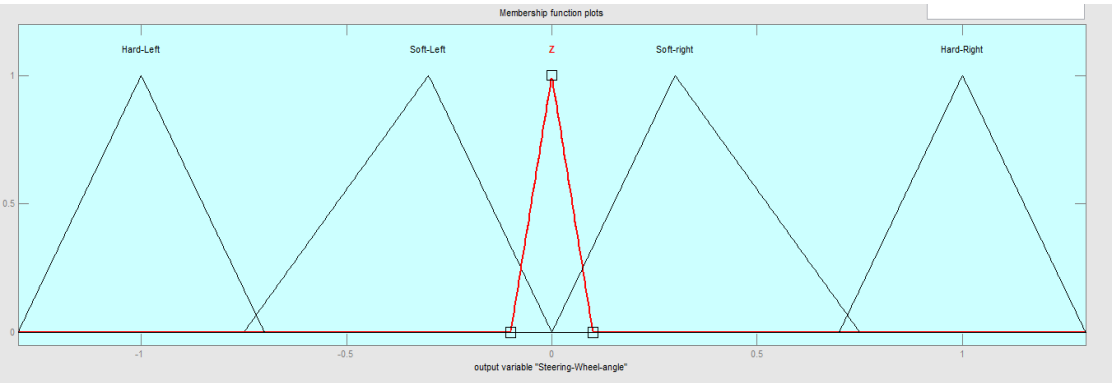


FIGURE 3.5.6: MATLAB 'Steering wheel angle' output membership function schematic.

2.5.3.5 Velocity membership function

For the velocity membership function, we will assign three (3) linguistic variables: Slow, Fast, and very Fast.

The corresponding membership function schematic is described in 0.

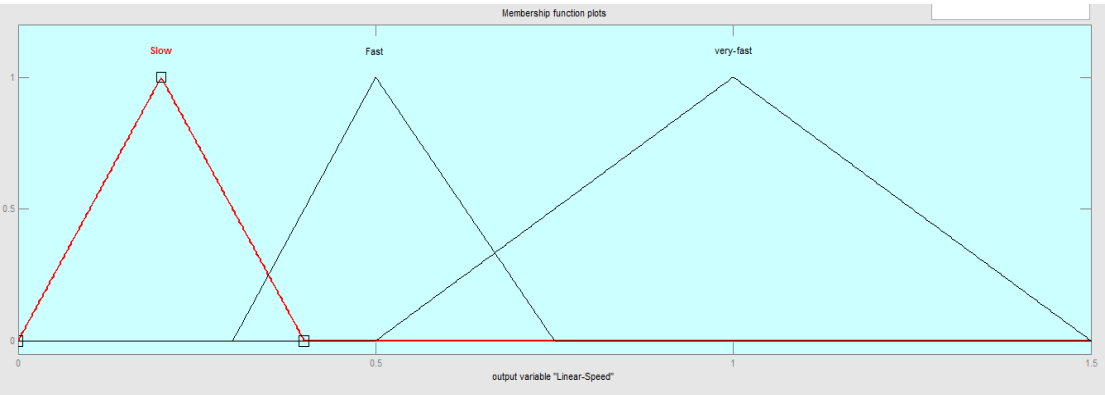


FIGURE 3.5.6: MATLAB 'Speed' output membership function schematic.

2.5.3.6 Inference engine

The inference engine is constructed of an "if distance is d_e and Orientation is θ_e and Change in Orientation is $\dot{\theta}_e$, then velocity is v and the steering wheel angle is β" statements. In order to receive a crisp output value we will use the "Center of mass" technique.

The inference engine correlation between the orientation and the change in orientation is described in Table 3.5.1 below.

Table 3.5.1 – Inference table between θ_e and $\dot{\theta}_e$

$\dot{\theta}_e$ \ θ_e	BN	SN	Z	SP	BP
VN				R	Z
N			R	Z	L
Z	HR	R	Z	L	HL
P	R	Z	L	Z	
VP	Z	L			

Obstacle maneuvering

One of the benefits of using the fuzzy logic controller, is that there is an easy way to implement obstacle maneuvering by changing the weight of each rule in case of obstacle detection.

The detection of the obstacles is done using the LiDAR. The LiDAR provides a good mapping of the surrounding with minimum of calculation, as opposed to visual point cloud from the cameras.

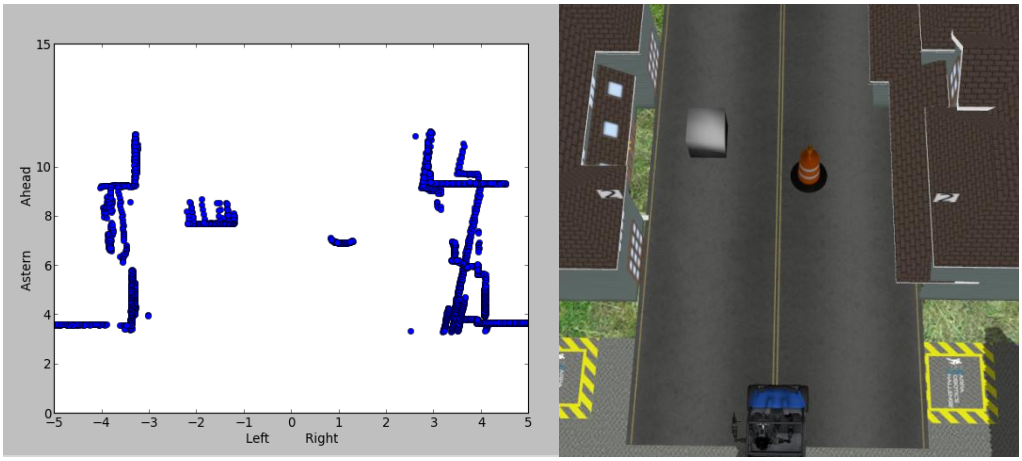


FIGURE 3.5.7: On the left 2D grid of the obstacles seen on the right.

The points are collected and per frame a new weight is calculated for each one of the orientation membership functions. The weight is calculated for each membership function depending on the proximity to the vehicle and amount of obstacle points in the region, as the proximity increases the weight decreases and the other membership function will have greater weight and the vehicle will diverge from the obstacle.

2.5.3.7 Procedures

2.5.3.7.1 ROS 'actionlib'

The high-level driving task is written as a ROS actionlib. Actionlib is used for monitoring and the ability to stop the script after it is executed, as opposed to ROS service-client.

2.5.3.7.2 Receiving way points from path planning

The first operation is to collect the way points from the path planner, these points are sent via a ROS "topic", and recollected each time 50% of the points in the buffer are reached. These points act as a guide line, and it is not mandatory to pass each one of them, or even to pass them very close.

2.5.3.7.3 Calculating the desired speed and steering wheel angle

Once we have the way points, we can start the logical driving process. The way point is used to send data to the FLC, along with the weight factor calculated using the LiDAR. As mentioned before, the output of the FLC is the desired speed and steering wheel angle.

2.5.3.7.4 Sending the calculated speed and steering angle

Dexterity will collect the values sent to them, via a ROS "service" and perform the actual movement by Atlas.

2.5.4 Results

After egression and first car initialization (gripping the steering wheel, moving gear to forward, placing the feet on the pedals and releasing the hand brake), the ROS 'actionlib' is started. Atlas starts driving to the received way points until it reaches the last gate. There it stops with a 'success' result.

The driving task is performed on an 8 meter wide road and consisted of a starting gate, end gate and obstacles. 0 shows the route driven by Atlas. The route is approximately 270 meters long.

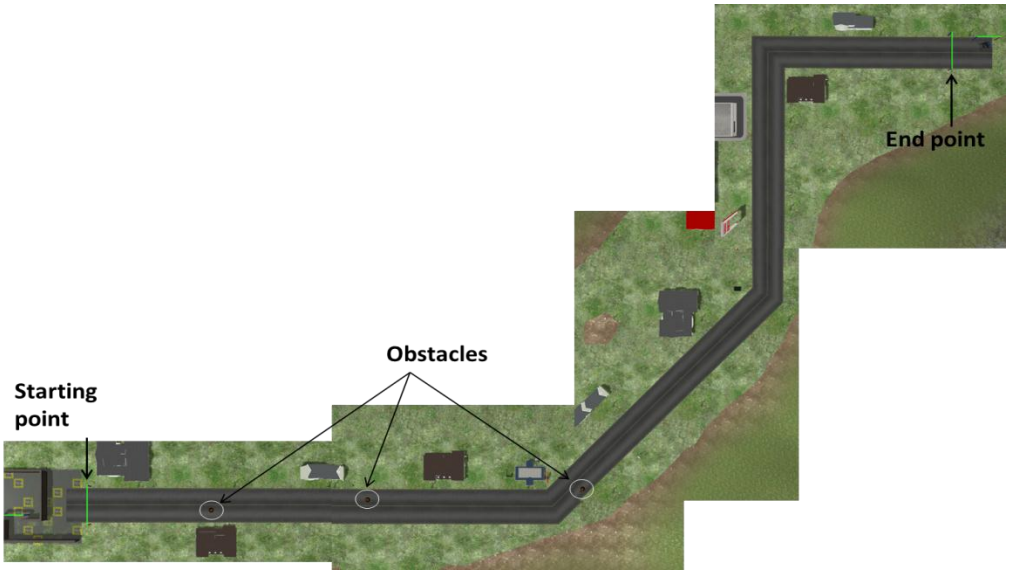


FIGURE 3.5.8: The route driven by Atlas.

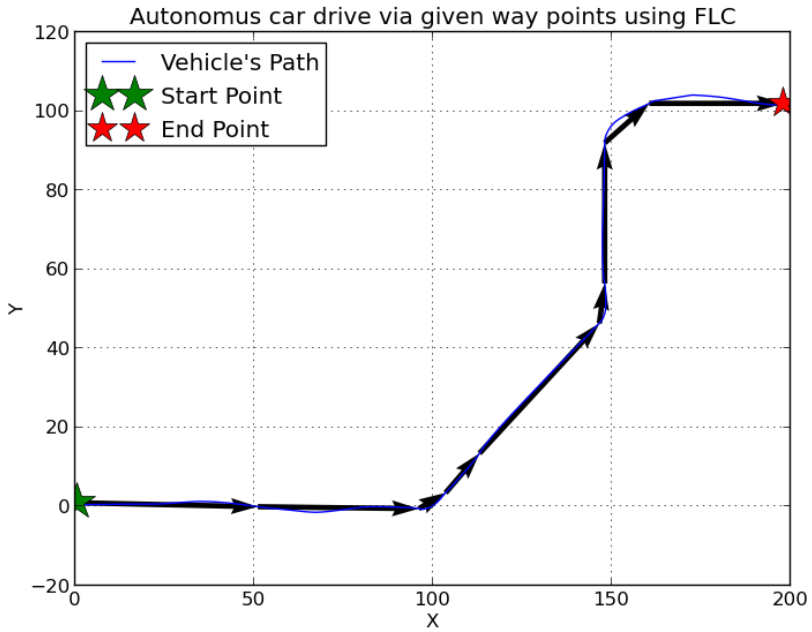


FIGURE 3.5.9: Atlas' path for driving task.

Atlas' path is recorded and shown in 0. You can view the results in a movie uploaded to YouTube. In the movie you can see Atlas handling the vehicle in the three stages of driving. In the first stage, the initialization stage, Atlas releases the handbrake, and grasps the hand wheel. In the second stage, the driving stage, Atlas drives the utility car, while maneuvering obstacles, from the starting point until the end point, which is the last gate. The third stage (which is not shown in the movie), the finalization stage, Atlas pulls the handbrake, and releases the hand wheel.

2.5.5 Discussion

The excellent way point tracking can be seen in 0. In addition, one can see the obstacle avoidance performed before the barrels (30 and 60 meters from the starting gate).

The fuzzy logic controller is an excellent tool for the driving task since the car's parameters are unknown to Atlas, and yet the FLC doesn't need these parameters in order to operate with good performance. In addition, the FLC's membership functions and values can be tuned very easily and can even be adaptive in some more complicated algorithms.

2.5.6 Conclusions

A fuzzy logic controller is a very simple, yet robust, way to design control systems using human experience over mathematical modeling. Moreover, it is a good way for bypassing linearization for nonlinear systems, thus making it very suitable for this type of control loops since the car's parameters are unknown. In addition, an obstacle maneuvering algorithm is quite easy to implement, a task which is hard, or even impossible, to implement in classic control systems.

2.6 Dexterity

2.6.1 Dexterity problem description

- Manipulate the steering wheel, car ignition, parking and driving modes.
- Hose/cable connection task.
- Manipulate Driller.

2.6.2 Task: Single arm motion

Introduction:

This module receives a position and orientation destination vector and returns a joint angles vector which bring the arm to the desired position.

Method:

A single arm motion is attained as combination of successive changes of the arms six joints. In order to perform the motion, a relation between position/orientation to the arm joints angles needs to be set. The solution can be attained by employing Inverse Kinematics approach which finds a specific solution (analytic or numeric) for the 6 equations created by Forward Kinematics representation (3.6.1).

$$F.K: \begin{pmatrix} x & y & z \\ roll & pitch & yaw \end{pmatrix} = f(q_1 \quad q_2 \quad q_3 \quad q_4 \quad q_5 \quad q_6)$$

$$I.K: \begin{pmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \\ q_6 \end{pmatrix} = f^{-1} \begin{pmatrix} x & y & z \\ roll & pitch & yaw \end{pmatrix} \qquad (3.6.1)$$

The resulted 6 joints angles obtained is input to the joints position PID controllers, so the arm moves directly to its goal.

2.6.3 Single arm motion and numerical solution

Solving the inverse kinematics problem of the hand relative to the shoulder gives us six equations for positions and orientation. The solution gives us angles of the six joints of the arm. This means that we got 6 nonlinear equation and 6 parameters to find.

We made an inverse kinematics solver for the hand relatively to the shoulder. We changed the solution so it considers the orientation inside its calculation. To generate a solution a Jacobian of the forward kinematics has been calculated to describe infinitesimally velocity. Integrating the velocity results an infinitesimally position. This numeric method applied to create a velocity and position profiles so at the end it gives the total Inverse Kinematics solution.

$$Q_{arm}[i+1] = \int J[i] \cdot \{\bar{V}_{global}\} dt \quad (3.6.2)$$

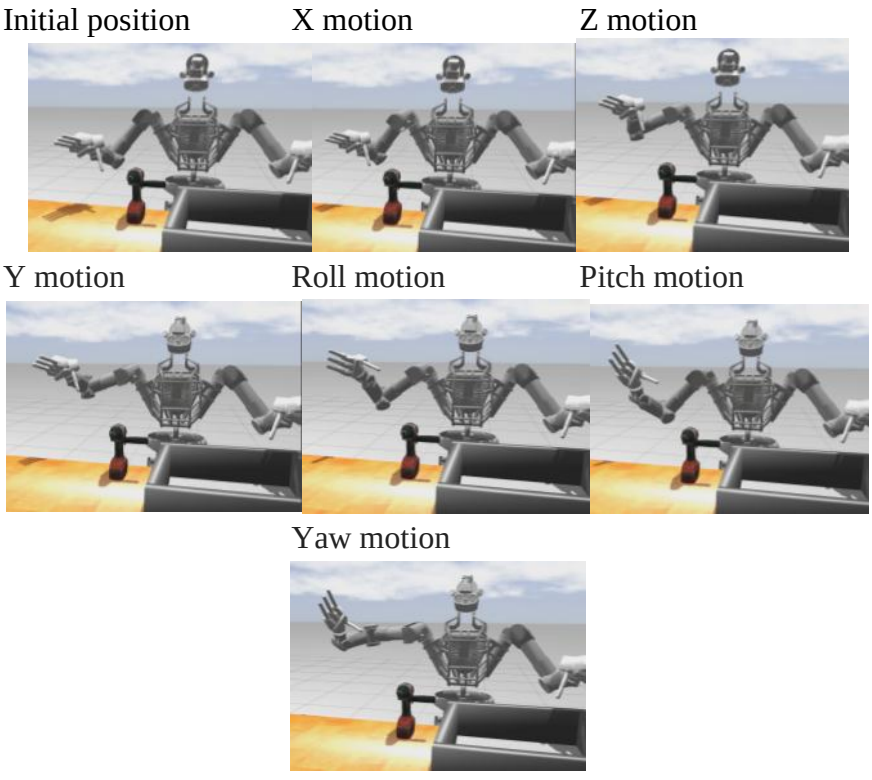


FIGURE 3.6.1: Robils Hand motion in x,y,z,roll,pitch,yaw directions.

For grasping object (like in the driller qualification) a power grasp is obtained and is being implemented such a way it close the fingers from an initial position until the force sensor stops the process.

2.6.4 Single Arm motion Analytic method

2.6.4.1 Forward Kinematics

The last 6 degrees of Freedom from shoulder to hand form the arm movement without the fingers. The placement of the hand is driven also from the 3 back joints. The base position is the pelvis position.We number the joints from the pelvis to the hand q1-q9.

Transformation Matrices

2.6.4.1.1 Back joints

Back links are driven by joints

- Lower Torso (ltorso) is turned around Z axis by the back_lbz joint.(q1)
- Middle Torso (mtorso) is turned around Y axis by the back_mby joint.(q2)
- Upper Torso (utorso) is turned around X axis by the back_ubx joint.(q3)

The Transformation matrices are:

$$ {}^{0}T_{1}=\begin{bmatrix} C\theta_1 & -S\theta_1 & 0 & d_{x1} \\ S\theta_1 & C\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^{1}T_{2}=\begin{bmatrix} C\theta_2 & 0 & S\theta_2 & 0 \\ 0 & 1 & 0 & 0 \\ -S\theta_2 & 0 & C\theta_2 & d_{z2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^{2}T_{3}=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C\theta_3 & -S\theta_3 & 0 \\ 0 & S\theta_3 & C\theta_3 & d_{z3} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

(3.6.3)

These transformation matrices are used for both arms equally.

2.6.4.1.2 Arm Joints

We have 6 joints in the arm.

- Upper Shoulder (clav) is turned around 2 axis Y and Z by the right/ left arm_usy joint (q4 right/left)
- Shoulder (scap) is turned around X axis by the right/left arm_shy joint (q5 right/left)
- Upper Arm (uram) is turned around Y axis by the right/left arm_ely (q6 right/left)
- Lower Arm (larm) is turned around X axis by the right/left arm_elx (q7 right/left)
- Forward Arm (farm) is turned around Y axis by the right/left arm_umy (q8 right/left)
- Hand (hand) is turned around X axis by the right/left arm_mwx (q9 right/left)

The Transformation Matrix of the Upper Shoulder (q4) is the most complex:

$$ {}^{3}T_{4}=\begin{bmatrix} C\theta_4 & -r_{z4}S\theta_4 & r_{y4}S\theta_4 & d_{x4} \\ r_{z4}S\theta_4 & r_{y4}{}^{2}V\theta_4+C\theta_4 & r_{y4}r_{z4}V\theta_4 & d_{y4} \\ -r_{y4}S\theta_4 & r_{y4}r_{z4}V\theta_4 & r_{z4}{}^{2}V\theta_4+C\theta_4 & d_{z4} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

(3.6.4)

$$ V\theta_i = 1 - Cos(\theta_i) \quad (3.6.5) $$

Transformation matrices of the rest of the joints (q5-q9)

$$ {}^{4}T_{5}=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C\theta_5 & -S\theta_5 & d_{y5} \\ 0 & S\theta_5 & C\theta_5 & d_{z5} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^{5}T_{6}=\begin{bmatrix} C\theta_6 & 0 & S\theta_6 & 0 \\ 0 & 1 & 0 & d_{y6} \\ -S\theta_6 & 0 & C\theta_6 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^{6}T_{7}=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C\theta_7 & -S\theta_7 & d_{y7} \\ 0 & S\theta_7 & C\theta_7 & d_{z7} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{7}T_{8}=\begin{bmatrix} C\theta_8 & 0 & S\theta_8 & 0 \\ 0 & 1 & 0 & d_{y8} \\ -S\theta_8 & 0 & C\theta_8 & d_{z8} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^{8}T_{9}=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C\theta_9 & -S\theta_9 & d_{y9} \\ 0 & S\theta_9 & C\theta_9 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

(3.6.6)

Here is the table of constants for the right arm:

Axes System number	Link name	Joints	Joints angle	Axes constants
1	ltorso	back_lbz	θ_1	$d_{x1}=-0.0125$
2	mtorso	back_mby	θ_2	$d_{z2}=0.09$
3	utorso	back_ubx	θ_3	$d_{z3}=0.05$
4	r_clav	r_arm_usy	θ_4	$r_{y4}=-0.5, r_{z4}=0.866, d_{x4}=0.024, d_{y4}=-0.221, d_{z4}=0.289$
5	r_scap	r_arm_shx	θ_5	$d_{y5}=-0.075, d_{z5}=0.036$
6	r_uram	r_arm_ely	θ_6	$d_{y6}=-0.185$
7	r_larm	r_arm_elx	θ_7	$d_{y7}=-0.121, d_{z7}=0.013$
8	r_farm	r_arm_umy	θ_8	$d_{y8}=-0.188, d_{z8}=-0.013$
9	r_hand	r_arm_mwx	θ_9	$d_{y9}=-0.058$

2.6.4.2 Inverse Kinematics

If We Know the 3 back joints position, we will calculate the rest 6 joint position by the give hand position and orientation.

2.6.4.2.1 The q4 joint problem

The Upper Shoulder (q4) joint's Transformation is much more complicated than the rest of the joint's Transformation (see Forward Kinematics). While in the Forward Kinematics it's just a little more complex Matrix multiply, the Inverse Kinematics Solution turn to be impossibly complex. Over Idea was to scan the q4 joint, then calculate the rest 5 joints analytically.

2.6.4.2.2 Solving q5-q9 analytically

The Transformation matrix from q4 to q9:

$$ {}^{4}T_{9} = {}^{4}T_{5}\cdot{}^{5}T_{6}\cdot{}^{6}T_{7}\cdot{}^{7}T_{8}\cdot{}^{8}T_{9} = \begin{bmatrix} n & s & a & p \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6.7} $$

By inverse multiply from both sides we can get:

$$ {}^{5}T_{6}\cdot{}^{6}T_{7}\cdot{}^{7}T_{8} = {}^{4}T_{5}^{-1}\cdot\begin{bmatrix} n & s & a & p \\ 0 & 0 & 0 & 1 \end{bmatrix}\cdot{}^{8}T_{9}^{-1} \tag{3.6.8} $$

The multiplication result:

$$ \begin{bmatrix} c6\cdot c8 - c7\cdot s6\cdot s8 & s6\cdot s7 & c6\cdot s8 + c7\cdot c8\cdot s6 & z7\cdot s6 + z8\cdot c7\cdot s6 + y8\cdot s6\cdot s7 \\ s7\cdot s8 & c7 & -c8\cdot s7 & y6 + y7 + y8\cdot c7 - z8\cdot s7 \\ -c8\cdot s6 - c6\cdot c7\cdot s8 & c6\cdot s7 & c6\cdot c7\cdot c8 - s6\cdot s8 & z7\cdot c6 + z8\cdot c6\cdot c7 + y8\cdot c6\cdot s7 \\ 0 & 0 & 0 & 1 \end{bmatrix} = $$

$$ \begin{bmatrix} nx & sx\cdot c9 - ax\cdot s9 & ax\cdot c9 + sx\cdot s9 & \\ ny\cdot c5 + nz\cdot s5 & c9\cdot(sy\cdot c5 + sz\cdot s5) - s9\cdot(ay\cdot c5 + az\cdot s5) & s9\cdot(sy\cdot c5 + sz\cdot s5) + c9\cdot(ay\cdot c5 + az\cdot s5) & \\ nz\cdot c5 - ny\cdot s5 & c9\cdot(sz\cdot c5 - sy\cdot s5) - s9\cdot(az\cdot c5 - ay\cdot s5) & s9\cdot(sz\cdot c5 - sy\cdot s5) + c9\cdot(az\cdot c5 - ay\cdot s5) & \\ 0 & 0 & 0 & \end{bmatrix} $$

$$ \begin{matrix} px + ax\cdot y9\cdot s9 - sx\cdot y9\cdot c9 \\ py\cdot c5 - y5\cdot c5 + pz\cdot s5 - z5\cdot s5 + y9\cdot s9\cdot(ay\cdot c5 + az\cdot s5) - y9\cdot c9\cdot(sy\cdot c5 + sz\cdot s5) \\ pz\cdot c5 - z5\cdot c5 - py\cdot s5 + y5\cdot s5 + y9\cdot s9\cdot(az\cdot c5 - ay\cdot s5) - y9\cdot c9\cdot(sz\cdot c5 - sy\cdot s5) \\ 1 \end{matrix} \Big](1) \tag{3.6.9} $$

If we will use column 2 and 4 we will get 6 equations that don't involve q8

For column 2(orientation):

$$ \begin{pmatrix} s6\cdot s7 \\ c7 \\ c6\cdot s7 \end{pmatrix} = \begin{pmatrix} sx\cdot c9 - ax\cdot s9 \\ c9\cdot(sy\cdot c5 + sz\cdot s5) - s9\cdot(ay\cdot c5 + az\cdot s5) \\ c9\cdot(sz\cdot c5 - sy\cdot s5) - s9\cdot(az\cdot c5 - ay\cdot s5) \end{pmatrix}(2) \tag{3.6.10} $$

For column 4(position):

$$\begin{pmatrix} z7\cdot s6+ z8\cdot c7\cdot s6+ y8\cdot s6\cdot s7 \\ y6+ y7+ y8\cdot c7- z8\cdot s7 \\ z7\cdot c6+ z8\cdot c6\cdot c7+ y8\cdot c6\cdot s7 \end{pmatrix} = \begin{pmatrix} px+ ax\cdot y9\cdot s9- sx\cdot y9\cdot c9 \\ py\cdot c5- y5\cdot c5+ pz\cdot s5- z5\cdot s5+ y9\cdot s9\cdot(ay\cdot c5+ az\cdot s5)- y9\cdot c9\cdot(sy\cdot c5+ sz\cdot s5) \\ pz\cdot c5- z5\cdot c5- py\cdot s5+ y5\cdot s5+ y9\cdot s9\cdot(az\cdot c5- ay\cdot s5)- y9\cdot c9\cdot(sz\cdot c5- sy\cdot s5) \end{pmatrix} (3)$$

(3.6.11)

Using (1) and (2) we can write (3) as:

$$\begin{pmatrix} z7\cdot s6+ z8\cdot c7\cdot s6+ y8\cdot s6\cdot s7 \\ y6+ y7+ y8\cdot c7- z8\cdot s7 \\ z7\cdot c6+ z8\cdot c6\cdot c7+ y8\cdot c6\cdot s7 \end{pmatrix} = \begin{pmatrix} px- s6\cdot s7\cdot y9 \\ c5\cdot(py- y5)- c7\cdot y9+ s5\cdot(pz- z5) \\ c5\cdot(pz- z5)- s5\cdot(py- y5)- c6\cdot s7\cdot y9 \end{pmatrix} (4)$$

(3.6.12)

After collecting arguments and moving expressions involving q6 and q7 from right side to left, we get:

$$\begin{pmatrix} s6\cdot(z7+ c7\cdot z8+ s7\cdot y89) \\ y67+ c7\cdot y89- s7\cdot z8 \\ c6\cdot(z7+ c7\cdot z8+ s7\cdot y89) \end{pmatrix} = \begin{pmatrix} px \\ c5\cdot py5+ s5\cdot pz5 \\ c5\cdot pz5- s5\cdot py5 \end{pmatrix} (5)$$

(3.6.13)

Where:

$$y67= y6+ y7\ ,\ y89= y8+ y9\ ,\ py5= py- y5\ ,\ pz5= pz- z5 \quad (3.6.14)$$

If we square the middle expression (5y) we get:

$$c7^2\cdot y89^2- 2\cdot c7\cdot s7\cdot y89\cdot z8+ 2\cdot c7\cdot y67\cdot y89+ s7^2\cdot z8^2- 2\cdot s7\cdot y67\cdot z8+ y67^2 = c5^2\cdot py5^2+ 2\cdot c5\cdot s5\cdot py5\cdot pz5+ s5^2\cdot pz5^2 (6)$$

(3.6.15)

If we square and add (5x) and (5z) we get:

$$c7^2\cdot z8^2+ 2\cdot c7\cdot s7\cdot y89\cdot z8+ 2\cdot c7\cdot z7\cdot z8+ s7^2\cdot y89^2+ 2\cdot y89\cdot s7\cdot z7+ z7^2 = c5^2\cdot pz5^2- 2\cdot c5\cdot S5\cdot py5\cdot pz5+ s5^2\cdot py5^2+ px^2 (7)$$

(3.6.16)

Adding (6) and (7):

$$y67^2+ 2\cdot c7\cdot y67\cdot y89- 2\cdot s7\cdot y67\cdot z8+ 2\cdot s7\cdot y89\cdot z7+ y89^2+ z7^2+ 2\cdot c7\cdot z7\cdot z8+ z8^2 = px^2+ py5^2+ pz5^2 (8)$$

(3.6.17)

Collecting and moving all arguments to left side:

$$c7\cdot 2\cdot(y67\cdot y89+ z7\cdot z8)+ s7\cdot 2\cdot(y89\cdot z7- y67\cdot z8)- px^2- py5^2- pz5^2+ y67^2+ y89^2+ z7^2+ z8^2 = 0 (9)$$

(3.6.18)

We can represent the equation as:

$$c7\cdot cc+ s7\cdot cs+ c1= 0 (10) \quad (3.6.19)$$

If we use this transformation:

$$\sin(q)= \frac{2\cdot x}{1+ x^2} \qquad \cos(q)= \frac{1- x^2}{1+ x^2} \qquad x= \tan\left(\frac{q}{2}\right) \quad (3.6.20)$$

We get a square equation:

(3.6.21) $$cc \cdot (1 - x7^2) + cs \cdot 2 \cdot x7 + c1 \cdot (1 + x7^2) = 0 \ (11)$$

The 2 solutions are:

(3.6.22) $$x = \frac{-cs \pm \sqrt{(-c1^2 + cc^2 + cs^2)}}{c1 - cc} (12)$$

The q7 solution will be:

(3.6.23) $$q7 = atan2(2 \cdot x, 1 - x^2) (13)$$

Using q7 solution and (5y) we can get:

(3.6.24) $$c5 \cdot py5 + s5 \cdot pz5 - y67 - y89 \cdot c7 + z8 \cdot s7 = 0 (14)$$

We can use manipulation done in (10) – (13) to get q5.

Dividing (5x) by (5z) we get 2 options:

If the expression

(3.6.25) $$(z7 + c7 \cdot z8 + s7 \cdot y89) > 0$$

Is true, then:

(3.6.26) $$q6 = atan2(px, pz5 \cdot c5 - py5 \cdot s5) (15)$$

If not then:

(3.6.27) $$q6 = atan2(-px, -(pz5 \cdot c5 - py5 \cdot s5)) (16)$$

Using (2x) we get:

(3.6.28) $$c9 \cdot sx - s9 \cdot ax - s6 \cdot s7 = 0 (17)$$

We can use manipulation done in (10) – (13) to get q9.

We can divide (1[2,1]) by (1[2,3]) to get q8

If s7>0 then:

(3.6.29) $$q8 = atan2(ny \cdot c5 + nz \cdot s5, -s9 \cdot (sy \cdot c5 + sz \cdot s5) - c9 \cdot (ay \cdot c5 + az \cdot s5)) (18)$$

If not:

(3.6.30) $$q8 = atan2(-(ny \cdot c5 + nz \cdot s5), s9 \cdot (sy \cdot c5 + sz \cdot s5) + c9 \cdot (ay \cdot c5 + az \cdot s5)) (19)$$

We had here 3 square equations that will give as 8 solutions; from these 8 solutions we will find the right solution by calculation of error from the wanted position and orientation.

2.6.5 Joint coordinate systems

To create a common language which all the modules will communicate throw we define the pelvis as the major link. Because our numeric method inverse and forward kinematics works from the shoulder to the hand we had to perform a manipulation to integrate with the input (IK from pelvis to hand).

We define now the solution results from a simple matrix manipulation:

$$T_{Hand}^{shoulder} = \left(T_{Hand}^{Pelvis}\right)^{-1} \cdot T_{Shoulder}^{Pelvis}$$

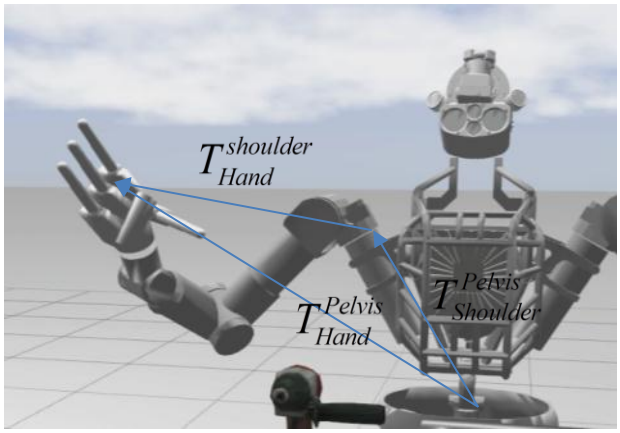


FIGURE 3.6.2: T shoulder - hand

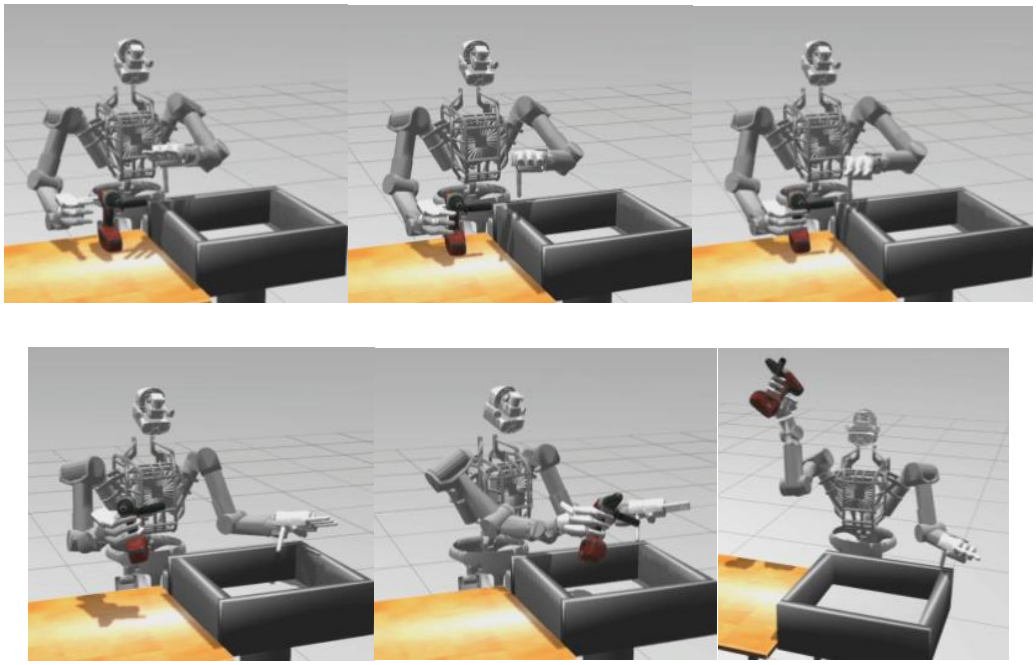


FIGURE 3.6.3: Object grasping and motion

2.6.6 Task: Integrating Dexterity with Vision

Description:

The main problem of this integration is to find the position and orientation of an object relative to the pelvis. In order to find it an object's position and orientation which the robot is capable to locate and grasp (with single hand) was pre-defined. After this zero position was set the robots vision could find a transformation (rotation and translation) from the zero position to its current position. This transformation matrix was manipulated on the hand's initial inverse kinematics such that the new solution brings the hand to grasp exactly on the same place on the object zero position.

Method:

1) Pre-Definition of the zero position of the Hose (Dexterity side):

Method of trial and error was implemented such a way we created a Task 3 world and launch files that brings up the robot stand next to the hose. After the simulation start up a sequence of IK points brought the hand to grasp the hose. The last point defines the "zero position". At this stage the vision should detect the hose so that it returns the identity matrix when asked for the position of the hose.

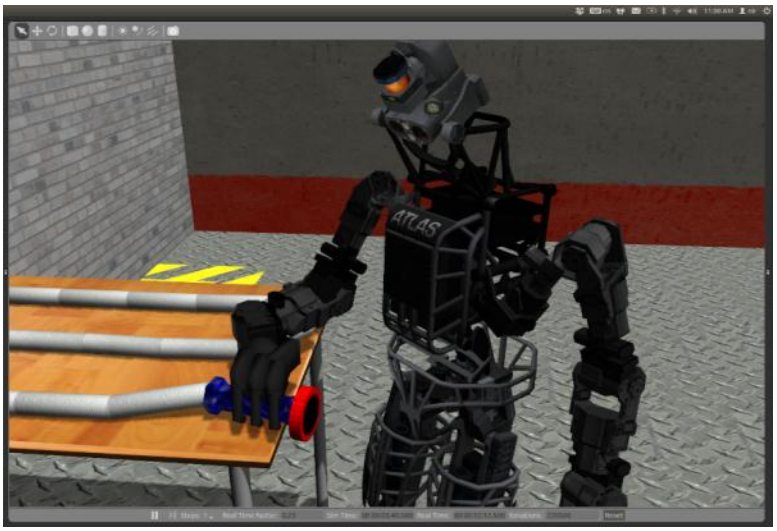


FIGURE 3.6.4: Zero position definition

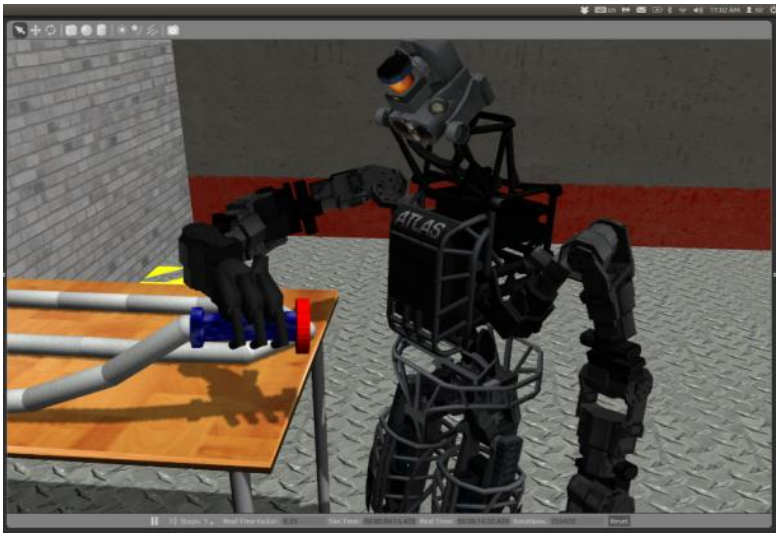


FIGURE 3.6.5: Lifting from zero position

2) Definition of un-known position of the hose:

In an unknown situation the vision knows to send transformation from the initial position to the hose current position in pelvis coordinate. The dexterity modules should multiply the returned matrix with the left side of the matrix generated from arm zero position (constant matrix). Using straight forward equation we can get the position and orientation and send them to the action modules.

$$T_{Hand}^{Pelvis} = T_{Current\ position}^{position\ 0} \cdot \left[T_{Hand}^{Pelvis} \right]_{position\ 0}$$

3 Conclusions

Looking back at the last 9 months, ROBIL group has transformed from 11 PI's and 35 students coming from 5 different institutes, into 1 united group, focused on autonomous robotics development. Apart from the administrative difficulty, we faced hard challenges regarding movement control, perception, decision making, and integration in a challenging environment, a parallel simulator development and in a very short time. Although we didn't have the chance to demonstrate all our developed capabilities during the VRC, we have made a huge step forward and was not too far from performing equally to the winning groups. We believe that given another 3-4 more weeks of efficient work we could demonstrate all of our planned capabilities. Concluding this, ROBIL will surely facilitate the spirit of DARPA and this challenge to pursuit autonomous abilities of humanoid robots control and behavior, and to become a global leader in advanced robotic software and hardware development.

4 References

1. S. Shimoda, Y. Kuroda and K. Iagnemma, "Potential Field Navigation of High Speed Unmanned Ground Vehicles on Uneven Terrain", Proceedings of the 2005 IEEE International Conference on Robotics and Automation.
2. Kim, Jonghoek, Fumin Zhang, and Magnus Egerstedt. "An exploration strategy by constructing Voronoi Diagrams with provable completeness." Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on. IEEE, 2009.
3. Atanas Georviev and Peter K. Allen, Localization Methods for a Mobile Robot in Urban Environments IEEE Transactions on Robotics, Vol.20, No.5, October 2004, pp.851-864
4. Garrido, Santiago, et al. "Path planning for mobile robot navigation using Voronoi diagram and fast marching." International Journal of Robotics and Automation (IJRA) 2.1 (2011): 42.
5. Colvin, Robert, and Ian J. Hayes. "A semantics for Behavior Trees." ARC Centre for Complex Systems, ACCS Technical Report ACCS-TR-07-01 (2007).
6. Kaminka, Gal A., et al. "Towards collaborative task and team maintenance" Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems. ACM, 2007.
7. Lim, Chong-U., Robin Baumgarten, and Simon Colton. "Evolving Behaviour Trees for the Commercial Game DEFCON."
8. Grunske, Lars, et al. "An automated failure mode and effect analysis based on high-level design specification with behavior trees." Integrated Formal Methods. Springer Berlin/Heidelberg, 2005.
9. D. E. Goldberg. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989. ISBN 0201157675.
10. Cheng M. Y. and C. S. Lin. Genetic algorithm for control design of biped locomotion. J. Robotic Systems, 14(5):365–373, 1997.
11. Flash T, Hogan N (1985) The coordination of arm movements: an experimentally confirmed mathematical model. J Neurosci 5:1688 –1703.
12. Hogan N. Impedance control: an approach for manipulation, I-III, Biological Cybernetics, 107:1-24, 1985.
13. A. J. Ijspeert, Central pattern generators for locomotion control in animals and robots: A review, *Neural Networks*, 21(4):642-653, 2008.
14. Kajita S. Kanehiro F. Kaneko K. Fujwara K. Harada K. Yukoi K. and Hirukawa H, Biped Walking Pattern Generation by using Preview Control of Zero Moment Point. Proc. IEEE IntConf Robotics & Automation, 2003
15. Kajita S, Morisawa M, Miura K, Nakaoka S, Harada K, Kaneko K, Kanehiro F and Yokoi K. Biped Walking Stabilization Based on Linear Inverted Pendulum Tracking. Proc. IEEE IntConf Intl Robots, 2010.
16. Klein TJ, Lewis MA (*2012)*. A physical model of sensorimotor interactions during locomotion. Neural Eng. 9 046011
17. A.D. Kuo, The Relative Roles of Feedforward and Feedback in the Control of Rhythmic Movements. *Motor Control, 2000* .

18. Lipson, H. *Evolutionary Robotics for legged Machines (2006): From Simulation to Physical Reality.* Computational Synthesis Lab Cornell University, Ithaca NY 14853, USA
19. Pratt J.E. and G. A. Pratt, Exploiting Natural Dynamics in the Control of a 3D Bipedal Walking Simulation, IntConf Climbing and Walking Robots (CLAWAR99), 1999
20. Spitz J, Y. Or, and M. Zacksenhouse. Towards a biologically inspired open loop controller for dynamic biped locomotion. In IEEE IntConf on Robotics and Biomimetics (ROBIO), 2011, pages 503–508, dec. 2011.
21. A J Ijspeert. A connectionist central pattern generator for the aquatic and terrestrial gaits of a simulated salamander. Biological Cybernetics, 84:331–348, 2001.
22. Thrun, Sebastian, et al. "Stanley: The robot that won the DARPA Grand Challenge." The 2005 DARPA Grand Challenge (2007): 1-43.
23. Kelly, A., & Stentz, A. (1998). *An Approach to Rough Terrain Autonomous Mobility.* Pittsburgh, PA: Robotics Institute, Carnegie Mellon University.
24. Driankov, D., Hellendoorn, H., Reinfrank, M., "An Introduction to Fuzzy Control", Springer Verlag, 1993.
25. Pawlikowski, Scott, "Development of a Fuzzy Logic Speed and Steering Control System For an Autonomous Vehicle", University of Cincinnati, 1999.
26. Lakkad, S. (2003). Modeling and Simulation of Steering Systems for Autonomous Vehicles. Florida State University.
27. Nguyen, H. T., Prasad, N. R., Walker, C. L., & Walker, E. A. (2003). A first course in fuzzy and neural control. New York: Chapman & Hall/CRC.
28. Pawlikowski, S. (1999). Development of a fuzzy logic speed and steering control system for an autonomous vehicle. University of Cincinnati.

5 List of Symbols, Abbreviations, and Acronyms

BDI: Belief-desired-intension software model

COM or CoM: Center off Mass

CPG: Central-Pattern Generator

FLC: Fuzzy Logic Controller

HMI: Human Machine Interface

HTN: Hierarchical Task Network

LiDAR: Light Detection and Ranging

OCD: Operational Concept Definition

OCU: Operator Control Unit

TLD: Track, Learn and Detect algorithm

VV&T: Verification, Validation and Testing

ZMP: Zero-Moment point

Symbols list

$x_d, y_d - Position\ of\ desired\ destination.$

$x_{car}, y_{car} - Position\ of\ the\ car$

$\theta_{car} - Car's\ azimuth$

$\dot{\theta}_{car} - Car's\ orientation\ rate$

$\theta_e - Error\ in\ orientation$

$d_e - Distance\ to\ way\ point$

$v - Velocity$

$\beta - Steering\ wheel\ angle$